Linear-Size Boolean Circuits for Multiselection

Justin Holmgren
NTT Research, Sunnyvale, CA, USA

Ron Rothblum
Technion, Haifa, Israel

Abstract

We study the circuit complexity of the *multiselection problem*: given an input string $x \in \{0,1\}^n$ along with indices $i_1, \ldots, i_q \in [n]$, output $(x_{i_1}, \ldots, x_{i_q})$. A trivial lower bound for the circuit size is the input length $n + q \cdot \log(n)$, but the straightforward construction has size $\Theta(q \cdot n)$.

Our main result is an $O(n + q \cdot \log^3(n))$-size and $O(\log(n+q))$-depth circuit for multiselection. In particular, for any $q \leq n/\log^3(n)$ the circuit has linear size and logarithmic depth. Prior to our work no linear-size circuit for multiselection was known for *any* $q = \omega(1)$ and regardless of depth.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Private Information Retrieval, Batch Selection, Boolean Circuits

Digital Object Identifier 10.4230/LIPIcs.CCC.2024.11

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2023/113/

Funding *Ron Rothblum*: Ron Rothblum is funded by the European Union (ERC, FASTPROOF, 101041208). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements We thank Yuval Ishai for useful discussions and his encouragement and an anonymous reviewer for useful comments.

1 Introduction

In the *selection problem*, also commonly called multiplexing, one is given a long string $x \in \{0,1\}^n$ and an index $i \in [n]$, and the goal is to output x_i. Selection is one of the basic operations in the word RAM model (and indeed costs unit time in that model). Implementing it by a Boolean circuit however is more involved and requires a circuit of size $\Theta(n)$.[1]

In this work we investigate a natural generalization of selection, which we call *multiselection*; now rather than a single index, one is given indices $i_1, \ldots, i_q \in [n]$, and the goal is to compute $(x_{i_1}, \ldots, x_{i_q})$. We denote this function by $\mathsf{Sel}^{n \to q}$. Multiselection is trivial in the word RAM model of computation (it costs $\Theta(q)$ operations with any standard instruction set). We focus on implementing multiselection by small *Boolean circuits*, where here and throughout this work all circuits have bounded fan-in. In this setting, we do not have matching upper and lower bounds. The best circuit size lower bound for multiselection is the input length $n + q \cdot \log n$. On the other hand, the straightforward construction (which selects each index separately) is much larger, with size $\Theta(n \cdot q)$. A slightly more sophisticated approach reduces to sorting, and results in a circuit of size $\Theta(n \log^2 n)$ for any $q = O(n)$. Somewhat surprisingly, it appears that multiselection has not been systematically studied and in particular we are not aware of essentially any other upper bounds (see further discussion in Section 1.1).

[1] The $\Omega(n)$ lower bound simply comes from the input size (which is a circuit size lower bound for any function in which each input has non-zero influence). The upper bound follows from a simple divide and conquer approach (cf. [27, Lemma 2.5.5] or Proposition 2).

Our main result is an $O(n)$-size, and $O(\log n)$-depth, Boolean circuit for multiselection for any $q \leq O(n/\log^3(n))$. More generally: [Main Theorem] For all $n, q \in \mathbb{Z}^+$ there is a Boolean circuit computing $\mathsf{Sel}^{n\to q}$ of size $O(n + q \cdot \log^3(n))$ and depth $O(\log(n+q))$. To the best of our knowledge, no linear-size circuit for multiselection was previously known for *any* $q = \omega(1)$, even without any restriction on the depth.

The main technical tools used in our construction are self-routing superconcentrators [25] and low-depth quasi-linear size sorting networks [1]. We also discuss applications of our linear-size multiselection circuit to problems in cryptography in Appendix A.

▶ Remark 1 (Uniformity). The circuits that we construct to prove Section 1 are *log-space uniform*, meaning that they can be generated in logarithmic space (and polynomial-time) by a uniform algorithm, given 1^n as input. Motivated by some of our applications, in Appendix B we also show an extension of the result with a *linear-time* uniform generation algorithm (on a word RAM machine), but only for $q < n^\varepsilon$, where $\varepsilon > 0$ is a universal constant.

1.1 Related Work

1.1.1 Tight Compaction

Several recent works [5, 22, 4, 3] study the related and central problem of tight compaction, in which the task is to retrieve all "marked" symbols of a string $\mathbf{x}$ (in any order). Although this has a similar flavor to multiselection, there are several important differences. First, compaction circuits assume that each symbol x_i of $\mathbf{x}$ is marked with a bit b_i indicating whether or not x_i is to be selected. In multiselection, we are instead given a *list* of indices $i_1, \ldots, i_q$ that should be selected. Given such a list, we know how to compute $(b_1, \ldots, b_n)$ with circuits only of size $\Omega(n \log^2 n)$. Second, the ordering of a compaction circuit's output is unconstrained, while the ordering of a multiselection circuit's output is determined by the ordering of its input indices $i_1, \ldots, i_q$.

We address similar issues in our approach to constructing multiselection circuits. Interestingly, our construction implicitly utilizes a variant of compaction that can be instantiated in linear size using the [5] construction. We discuss this point further in our technical overview.

1.1.2 Uniselection Lower Bounds.

As noted above, it is easy to construct an $O(n)$-size circuit for the *uniselection* problem (i.e., the case of $q = 1$). Interestingly though, the uniselection problem has been a source for some of the best known circuit and formula lower bounds that are known (for any explicit function).

In particular, Paul [24] gave a $2n - o(n)$ lower bound on the circuit complexity of uniselection (over any 2-bit gate basis). Blum [7] gave a $3n - o(n)$ lower bound for computing a closely related function (as a matter of fact this function involves computing a multiselection with $q = 3$ and applying a simple gadget function to the result). Nechiporuk [23] proved a quadratic lower bound on the formula size for a modified selection function and Andreev [2] proved an $n^{2.5-o(1)}$ lower bound on the de Morgan formula size of roughly the same function.

2 Technical Overview

In this section we give an overview of the proof techniques underlying the proof of Section 1. In this overview we focus on achieving a *linear-size* multiselection circuit and defer the additional complications needed to simultaneously achieve *logarithmic-depth* to the technical sections.

The main technical tool that we use in our construction is a *superconcentrator*. Recall that a superconcentrator [28] is a constant-degree directed acyclic graph with $O(n)$ vertices, n of which are sources and n of which are sinks, with the property that for any $q \leq n$, any set S of sources, and any set T of sinks with $|S| = |T| = q$, there exist q vertex-disjoint paths from S to T.

2.1 Weak multiselection from superconcentrators

We first use superconcentrators to construct linear-size circuits for a *weak* form of multiselection (to be described shortly) and then show how to upgrade this construction to a circuit for full-fledged multiselection.

By replacing each vertex of a superconcentrator with a constant-sized routing gadget, we obtain for every $q \leq n$ a circuit $\hat{C}$ of size $O(n)$ that implements the following weak form of multiselection: For any list of distinct indices $\mathbf{i} = (i_1, \ldots, i_q) \in [n]^q$, there exists an "advice string" $\hat{\mathbf{i}} \in \{0,1\}^{\Theta(n)}$ and a reordering $j_1, \ldots, j_q$ of $i_1, \ldots, i_q$ such that $\hat{C}(\mathbf{x}, \hat{\mathbf{i}}) = (x_{j_1}, \ldots, x_{j_q})$, for all $\mathbf{x} \in \{0,1\}^n$. The string $\hat{\mathbf{i}}$ describes the q vertex-disjoint paths from the source vertices $i_1, i_2, \ldots, i_q$, to the sinks $1, 2, \ldots, q$, by specifying how each vertex's routing gadget should be configured.

This weak form of multiselection suffers from three drawbacks, which we will repair one by one:

1. The cost of computing $\hat{\mathbf{i}}$ might be large, and in particular super-linear in the length of $\mathbf{x}$,
2. The output of $\hat{C}$ is misordered – it is $(x_{j_1}, \ldots, x_{j_q})$ instead of $(x_{i_1}, \ldots, x_{i_q})$, and
3. The input $\mathbf{i}$ is required to consist of *distinct* indices in $[n]$.

2.2 Amortizing the computation of the advice string

We address issue 1 by (temporarily) switching to a larger alphabet, which allows to amortize the cost of computing the advice string $\hat{\mathbf{i}}$. We refer to the latter as a "block multiselector". Later we will show how to use a block multiselector to construct our desired (binary alphabet) multiselector.

In more detail, consider a generalization of multiselection, over an alphabet $\Sigma = \{0,1\}^s$. The input is now a string $\mathbf{x} \in \Sigma^n$ and indices $i_1, \ldots, i_q \in [n]$ and the goal, as before, is to output $(x_{i_1}, \ldots, x_{i_q}) \in \Sigma^q$. Jumping ahead, it will suffice for us to set the block size s to be poly-logarithmic in n.

To construct the block multiselector, denoted $\hat{C}_s$, we simply take s copies of $\hat{C}$ so that the j^{th} copy gets as input the j^{th} bit of each input block and produces the j^{th} bit of each output block; all copies of $\hat{C}$ share the same advice string input. This circuit $\hat{C}_s$ has the property that for all $\mathbf{i} \in [n]^q$ (consisting of distinct elements), there exists $\hat{\mathbf{i}} \in \{0,1\}^{\Theta(n)}$ and a reordering $\mathbf{j} = (j_1, \ldots, j_q)$ of $\mathbf{i}$ such that for all $x \in \left(\{0,1\}^s\right)^n$, it holds that $\hat{C}_s(x, \hat{\mathbf{i}}) = (x_{j_1}, \ldots, x_{j_q})$.

The size of $\hat{C}_s$ is $O(n \cdot s)$, but the computation of $\hat{\mathbf{i}}$ from $i_1, \ldots, i_q$ is entirely independent of s. Thus, by choosing s to be sufficiently large, we can hope to amortize the computation of $\hat{\mathbf{i}}$ relative to our input length, which is $\Omega(n \cdot s)$. There is an $O(n \log^2(n))$ circuit for computing $\hat{\mathbf{i}}$ for Pippenger's self-routing superconcentrators [25], so by using these in the prior step it suffices now to set $s = \log^2(n)$.

Overall, we obtain a linear-size block multiselector C_s (i.e., of size $O(n \cdot s)$), for block size $s = \log^2(n)$ and number of queries $q \leq n$, with the following property: for all distinct $i_1, \ldots, i_q \in [n]^q$, there exists a reordering $j_1, \ldots, j_q$ of $i_1, \ldots, i_q$ such that $C_s(\mathbf{x}, i_1, \ldots, i_q) = (x_{j_1}, \ldots, x_{j_q})$, for all $\mathbf{x} \in \left(\{0,1\}^s\right)^n$. We note that such a circuit could also be constructed using linear-size circuits for tight compaction [5], in conjunction with Lemma 21.

2.3 Reordering the outputs and handling non-unique inputs

We next simultaneously resolve issues 2 (misordered output) and 3 (the restriction of unique indices). The high level idea is to append to each of the blocks its (original) block index. This means that after applying the block-multiselector from the previous step, we still keep track of the original location of this block, which we can combine with $i_1, \ldots, i_q$ to rearrange the output blocks in the desired order (and handle multiplicities appropriately).

In more detail, on input $\mathbf{x} \in \left(\{0,1\}^s\right)^n$ and $\mathbf{i} \in [n]^q$, we construct the string $\mathbf{x}' := \big((1, x_1), \ldots, (n, x_n)\big) \in (\{0,1\}^{\log(n)+s})^n$ as well as an index string $\mathbf{i}' \in [n]^q$ that consists of a sequence of *distinct* elements that contains all elements of $\mathbf{i}$ (the string $\mathbf{i}'$ can be constructed by sorting $i_1, \ldots, i_q$, removing duplicates, and adding suitable padding). We then invoke $C_{s+\log n}$ on input $(\mathbf{x}', \mathbf{i}')$ to obtain the set

$$\big\{(i'_1, x_{i'_1}), \ldots, (i'_q, x_{i'_q})\big\}, \tag{1}$$

represented as a list of elements (in some order).

We next combine (1) with

$$\big\{(1, i_1), \ldots, (q, i_q)\big\} \tag{2}$$

to obtain

$$\big\{(1, i_1, x_{i_1}), \ldots, (q, i_q, x_{i_q})\big\} \tag{3}$$

using the same representations of sets. This step, which can be viewed as an *inner join* (cf the database literature), combines the two lists based on the common keys $i_1, \ldots, i_q$. It can be implemented in quasilinear size using sorting circuits. Once we have constructed the set in Equation (3) in some arbitrary order, we can sort its elements by their first coordinate to obtain the ordered list $\big((1, i_1, x_{i_1}), \ldots, (q, i_q, x_{i_q})\big)$, from which we can read off $(x_{i_1}, \ldots, x_{i_q})$.

The dominant costs in this construction arise from the usage of sorting circuits. In each instance, standard sorting circuits are applicable, with size $\tilde{O}(q) \cdot (s + \log n)$. If we use the sorting circuits of [1], then the $\tilde{O}(q)$ factor is in fact $O(q \log q)$, so (with $s = \log^2(n)$) it suffices to set $q \leq n/\log^3(n)$.

To summarize, we have now built a linear-size block-multiselection circuit for querying $q = n/\log^3(n)$ blocks of size $s = \log^2(n)$.

2.4 From block-multiselection back to bit-multiselection

Finally, we return to our original goal of multiselection over $\{0,1\}$. We do so as follows: given queries $i_1, \ldots, i_q$ to *bits* of a string x, we group the bits of x into s-bit blocks $B_1, \ldots, B_{n/s}$. We view each index i_j as consisting of a prefix $p_j \in [n/s]$ (specifying the block index) and a suffix $r_j \in [s]$ (specifying the internal index within the block). We apply the multiselection circuit of the previous step to obtain blocks $B_{p_1}, \ldots, B_{p_q}$ indexed by the $\log(n/s)$-bit prefixes $p_1, \ldots, p_q$ of $i_1, \ldots, i_q$. We then use q copies of a (uni-)selection circuit to obtain and output the r_j^{th} bit from each block B_{p_j}. The circuit implements the desired multiselection functionality and has size $O((n/s) \cdot s + q \cdot s) = O(n)$.

2.5 Achieving low-depth

A straightforward implementiation of our construction has depth polylog(n). Improving this to $O(\log n)$ requires significant care. For example, while we can use a logarithmic depth sorting network [1], that network uses abstract comparator gates. When working over

a large alphabet, the implementation of each comparator gate has super-constant depth (when implemented as a Boolean circuit), and so the resulting overall depth is ostensibly super-logarithmic. Nevertheless, we are able to provide suitable implementations of all of the gadgets so that the overall construction achieves logarithmic depth.

3 Preliminaries

We assume that the reader is familiar with the notion of a Boolean circuit and the associated function that it computes. We emphasize that, unless otherwise stated, throughout this work by "circuit" we refer to constant fan-in circuits over the standard De Morgan base.

3.1 Uniformity

We say that a family $\mathcal{C} = \{C_n\}_{n\in\mathbb{Z}^+}$ of Boolean circuits is *log-space uniform* if there exists a Turing machine that on input 1^n uses $O(\log n)$ space and prints the description of the circuit C_n, on a write-only output tape. The description is a listing of the gates in an (arbitrary) topological ordering along with a specification of the gate operation and pointers to each of its inputs. In Appendix B we also discuss an extension of our main result for *linear-time* uniformity.

3.2 Boolean Circuits for Uniselection

The following standard proposition gives a linear-size circuit, that given a string $\mathbf{x} \in \Sigma^n$, over an alphabet Σ, and an index $i \in [n]$, outputs the symbol x_i (in other words, a multiplexer).

▶ **Proposition 2.** *Let $s = s(n) \in \mathbb{Z}^+$ and let $\Sigma = \{0,1\}^s$. There exists a log-space uniform Boolean circuit for $\mathsf{Sel}_{\Sigma}^{n\to 1}$ with size $O(n \cdot s)$ and depth $O(\log n)$*

Proof. Without loss of generality assume that $s = 1$ (for $s > 1$ we can use s parallel copies of the circuit for $s = 1$). The circuit follows a divide and conquer strategy. Assume for simplicity that n is a power of 2. Let $\mathbf{x} \in \{0,1\}^n$ and $i \in [n]$ be the inputs. Let i_1 denote the most significant bit of i and let i' denote the remaining bits (i.e., $i' \in [n/2]$). To select the i^{th} bit of $\mathbf{x}$, the circuit recursively selects the $(i')^{th}$ bit of both the lower and upper halves of $\mathbf{x}$. Then, based on i_1, it decides which of the two bits to output. Overall the circuit size satisfies the recursion: $S(n) = 2S(n/2) + O(1)$ and the depth satisfies $D(n) = D(n/2) + O(1)$. The proposition follows. ◀

3.3 Small-Size Sorting Circuits

Our circuits for multiselection heavily rely on circuits for sorting integers.

▶ **Lemma 3.** *For every $k = k(n)$ and $m = m(n)$, there is a log-space uniform Boolean circuit of size $O(n \cdot m \cdot \log n)$ for sorting n integers of m bits.*

This lemma follows immediately from the sorting networks of Ajtai, Komlós, and Szemerédi [1]. In order to make our multiselection circuit have logarithmic depth, we need a refined version of Lemma 3, which we present in Section 4.

4 Low-Depth Sorting of Logarithmic-Length Keys

We first recall what it means to sort with respect to a partial ordering.

▶ **Definition 4.** *Let Σ be a finite set. We say that $\mathbf{x}, \mathbf{y} \in \Sigma^n$ are* reorderings *of each other if for some permutation $\pi : [n] \to [n]$, it holds that $x_i = y_{\pi(i)}$, for all $i \in [n]$.*

▶ **Definition 5.** *Let Σ be a finite set with a strict partial ordering $\prec$. A Boolean circuit is said to* sort Σ^n with respect to $\prec$ *if on any input $\mathbf{x} \in \Sigma^n$, it outputs a reordering $\mathbf{y}$ of $\mathbf{x}$ such that for any $i, j \in [n]$, $y_i \prec y_j \implies i < j$.*

In particular, we will focus on a partial ordering that compares integers by their k most significant bits. That is, for any $m \in \mathbb{Z}^+$ and any $\mathbf{x}, \mathbf{y} \in \{0,1\}^m$, we write $\mathbf{x} \prec_k \mathbf{y}$ to denote that $(x_1, \ldots, x_k)$ lexicographically precedes $(y_1, \ldots, y_k)$.

▶ **Lemma 6.** *For every $m = m(n)$ and $k \in [m]$, there is a log-space uniform Boolean circuit of size $O(n \cdot m \cdot \log n)$ and depth $O(k + \log n)$ for sorting $\left(\{0,1\}^m\right)^n$ with respect to $\prec_k$.*

We are mainly interested in the setting $k = \Theta(\log n)$, in which case the constructed Boolean circuit has size $O(n \cdot m \cdot \log n)$ and depth $O(\log n)$.

To the best of our knowledge Lemma 6 was not previously known. The straight-forward circuit based on AKS networks yields circuits with matching size $O(n \cdot m \cdot \log n)$ but depth $O(\log n \cdot \log k)$. Kospanov [17] obtained circuits with better depth $O(\log n + \log k)$ but much larger size $O(n^2 \cdot k)$. A recent work of [18] obtains size $O(n \cdot k^2)$ and depth $O(\log n + k \log k)$, which is better for some values of $k = o(\log n)$, but worse in our regime of $k = \Theta(\log n)$. The work of Lin and Shi [22, Theorem 1.1] similarly constructs circuits with size $O(n \cdot m \cdot \log n)$ and depth $O(\log n)$ when $n > 2^{4k+7}$, but not for all values of $k = \Theta(\log n)$.

4.1 Sorting Networks

Our sorting circuits rely on sorting *networks*, which are generic constructions of n-input sorting circuits from 2-input sorting circuits.

▶ **Definition 7** (Sorting Networks)**.** *An n-*input sorting network *is a circuit C with n inputs and n outputs, and with all gates having fan-in and fan-out two, such that for any set Σ with strict partial ordering $\prec$, if each gate of C is replaced by a circuit that sorts Σ^2 with respect to $\prec$, then the resulting circuit sorts Σ^n with respect to $\prec$.*

For the best asymptotic size and depth, we use the celebrated sorting network of Ajtai, Komlós, and Szemerédi [1].

▶ **Lemma 8** ([1])**.** *There is a (log-space uniform) n-input sorting network with size $O(n \log n)$ and depth $O(\log n)$.*

4.2 From Sorting Networks to Boolean Circuits

We prove Lemma 6 by combining the following lemma with Lemma 8.

▶ **Lemma 9.** *Suppose that there is a log-space uniform sorting network on n elements with size $S = S(n)$ and depth $D = D(n)$. Then, for all $m = m(n)$ and $k \in [m]$, there is a log-space uniform Boolean circuit with size $O(S \cdot m)$ and depth $O(D + k)$ for sorting $\left(\{0,1\}^m\right)^n$ with respect to $\prec_k$.*

Proof. Starting with an n-input sorting network $\mathcal{A}$ with size S and depth D, we obtain a Boolean circuit $\mathcal{C}$ from $\mathcal{A}$ by replacing each gate with a Boolean circuit $\mathsf{Sort}^{(2)}_{k,m}$ that sorts $\left(\{0,1\}^m\right)^2$ with respect to $\prec_k$. With this approach it is easy to make $\mathcal{C}$ have size $O(S \cdot m)$ – since $\mathsf{Sort}^{(2)}_{k,m}$ merely needs to have size $O(m)$.

Depth seems to pose more of a difficulty. Locality considerations imply that $\mathsf{Sort}^{(2)}_{k,m}$ must have depth at least $\Omega(\log k)$ (e.g. the least significant bit of either output of $\mathsf{Sort}^{(2)}_{k,m}(\mathbf{x}, \mathbf{y})$ depends on all of the k most significant bits of $\mathbf{x}$ and $\mathbf{y}$). Thus, it might seem that this approach dooms $\mathcal{C}$ to have depth $\Omega(D \cdot \log k)$.

We circumvent this difficulty by noting that although input-to-output paths in $\mathcal{C}$ are concatenations of D input-to-output paths in $\mathsf{Sort}^{(2)}_{k,m}$, these D paths cannot all be worst-case. In particular, the i^{th} output bit of one copy of $\mathsf{Sort}^{(2)}_{k,m}$ is only connected to the j^{th} input bit of another copy if $i \equiv j \pmod m$ (see Figure 1).

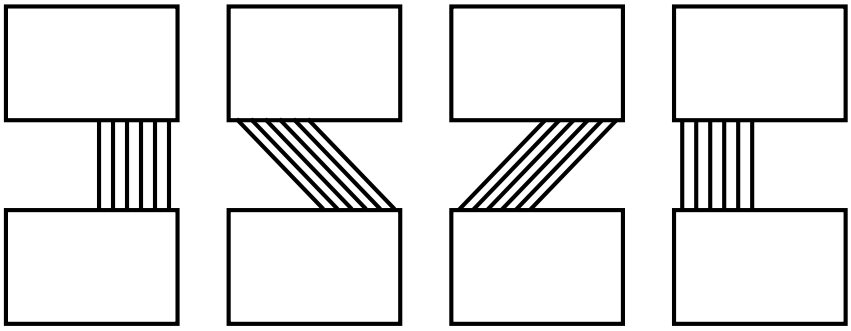

Figure 1 The four different ways that we can connect one copy of $\mathsf{Sort}^{(2)}_{k,n}$ to another. In this depiction, a copy of $\mathsf{Sort}^{(2)}_{k,n}$ is depicted by a rectangle, with inputs incident to the bottom edge and output incident to the top. The left half corresponds to the first input/output, and the right half to the second.

We leverage this with the following lemma, whose proof we defer momentarily.

▶ **Lemma 10.** *For any $m = m(n)$ and $k \in [m]$, there is a log-space uniform Boolean circuit $\mathsf{Sort}^{(2)}_{k,m}$ with size $O(m)$ that sorts $\left(\{0,1\}^m\right)^2$ with respect to $\prec_k$ and has the following additional property:*

For all $\hat{i}, \hat{j} \in [m]$ and $i, j \in [2m]$ with $i \equiv \hat{i} \pmod m$ and $j \equiv \hat{j} \pmod m$, every path in $\mathsf{Sort}^{(2)}_{k,m}$ from the i^{th} input vertex to the j^{th} output vertex has length $O\big(\min(j,k) - \min(i,k) + 1\big)$. In particular there is no such path if $k > i > j$.

Assuming Lemma 10, consider any input-to-output path $\mathbf{p}$ in $\mathcal{C}$. We write $\mathbf{p}$ as a composition of paths $\mathbf{p}_1 \circ \cdots \circ \mathbf{p}_D$, where each $\mathbf{p}_\ell$ is an input-to-output path of some copy of $\mathsf{Sort}^{(2)}_{k,m}$. Define $i_\ell, j_\ell \in [2m]$ so that $\mathbf{p}_\ell$ starts at the i^{th}_ℓ input and ends at the j^{th}_ℓ output of that copy, and define $\hat{i}_\ell, \hat{j}_\ell \in [m]$ such that $i_\ell \equiv \hat{i}_\ell \pmod m$ and $j_\ell \equiv \hat{j}_\ell \pmod m$. By the observation above, we have $\hat{j}_\ell = \hat{i}_{\ell+1}$ for all $\ell \in [D-1]$, and by Lemma 10, we have for some constant L that $|\mathbf{p}_\ell| \le L \cdot \big(\min(\hat{j}_\ell, k) - \min(\hat{i}_\ell, k) + 1\big)$.

Putting this together, we bound

$$
\begin{aligned}
|\mathbf{p}| &= \sum_{\ell=1}^{D} |\mathbf{p}_\ell| \\
&\le \sum_{\ell=1}^{D} L \cdot \big(\min(\hat{j}_\ell, k) - \min(\hat{i}_\ell, k) + 1\big) \\
&= L \cdot \big(\min(\hat{j}_D, k) - \min(\hat{i}_1, k) + D\big) && \text{(telescoping sum because } \hat{j}_\ell = \hat{i}_{\ell+1} \text{ for } \ell \in [D-1]\text{)} \\
&\le L \cdot (k + D) \\
&= O(k + D).
\end{aligned}
$$

◀

It remains to prove Lemma 10.

Proof of Lemma 10. We use the straightforward construction that compares the elements bit-by-bit starting with their most significant bits. In more detail, we will construct a circuit $\widetilde{\mathsf{Sort}}^{(2)}_{k,m} : \{\prec, ?, \succ\} \times \{0,1\}^m \times \{0,1\}^m$ that has the functionality

$$\widetilde{\mathsf{Sort}}^{(2)}_{k,m}(?, \mathbf{x}, \mathbf{y}) = \begin{cases} (\mathbf{x}, \mathbf{y}) & \text{if } \mathbf{x} \prec \mathbf{y} \\ (\mathbf{y}, \mathbf{x}) & \text{otherwise.} \end{cases}$$

$$\widetilde{\mathsf{Sort}}^{(2)}_{k,m}(\prec, \mathbf{x}, \mathbf{y}) = (\mathbf{x}, \mathbf{y})$$

$$\widetilde{\mathsf{Sort}}^{(2)}_{k,m}(\succ, \mathbf{x}, \mathbf{y}) = (\mathbf{y}, \mathbf{x})$$

and we define $\mathsf{Sort}^{(2)}_{k,m} = \widetilde{\mathsf{Sort}}^{(2)}_{k,m}(?, \cdot, \cdot)$. This clearly gives the desired functionality. The first input symbol aids in recursively constructing the circuit for $\widetilde{\mathsf{Sort}}^{(2)}_{k,m}$, and is meant to indicate whether the global decision on the comparison between x and y is (1) still undecided (?), (2) decided toward x or (3) decided toward y.

Our construction of $\widetilde{\mathsf{Sort}}^{(2)}_{k,m}$ is recursive. If $k = 0$, then we define $\widetilde{\mathsf{Sort}}^{(2)}_{k,m}$ to be the identity function. For $k > 0$, $\widetilde{\mathsf{Sort}}^{(2)}_{k,m}$ takes input $(c, \mathbf{x}, \mathbf{y})$, it first computes

$$(c', a_1, b_1) = \begin{cases} (\prec, x_1, y_1) & \text{if } c = \prec \text{ or } (c = ? \text{ and } x_1 < y_1) \\ (?, x_1, y_1) & \text{if } c = ? \text{ and } x_1 = y_1 \\ (\succ, y_1, x_1) & \text{if } c = \succ \text{ or } (c = ? \text{ and } x_1 > y_1). \end{cases}$$

Then, with $\mathbf{x}'$ and $\mathbf{y}'$ denoting $(x_2, \ldots, x_m)$ and $(y_2, \ldots, y_m)$, it computes

$$(\mathbf{a}', \mathbf{b}') = \widetilde{\mathsf{Sort}}^{(2)}_{k-1,m-1}(c', \mathbf{x}', \mathbf{y}').$$

Finally it outputs $\big(a_1 \circ \mathbf{a}', b_1 \circ \mathbf{b}'\big)$, where $\circ$ denotes string concatenation.

It is easy to check the correctness of this construction.

Circuit Size. Let $S(k, m)$ denote the size of $\widetilde{\mathsf{Sort}}^{(2)}_{k,m}$. For $k = 0$ we clearly have $S(k, m) = O(m)$. For larger k, the recursive construction gives $S(k, m) = S(k-1, m-1) + O(1)$, so by induction $S(k, m) = O(m)$ for all $k \leq m$.

Input-to-Output Path Lengths. We now bound the length of different input-to-output paths in $\widetilde{\mathsf{Sort}}^{(2)}_{k,m}$. As in the construction above, denote the input to $\widetilde{\mathsf{Sort}}^{(2)}_{k,m}$ by $(c, \mathbf{x}, \mathbf{y})$ and denote the output by $(\mathbf{a}, \mathbf{b})$.

▷ Claim 11. There exists a constant L such that any path in $\widetilde{\mathsf{Sort}}^{(2)}_{k,m}$ from an input $\{x_i, y_i\}$ to an output $\{a_j, b_j\}$ has length at most $L \cdot (j - i + 1)$, and any path from the input c to an output $\{a_j, b_j\}$ has length at most $L \cdot j$.

In particular any path to an output $\{a_j, b_j\}$ has length at most $L \cdot j$.

Proof. Let $(\mathbf{x}', \mathbf{y}')$ denote the input to the recursive call to $\widetilde{\mathsf{Sort}}^{(2)}_{k-1,m-1}$ and let $(\mathbf{a}', \mathbf{b}')$ denote the output of this call.

We consider several cases separately:

- Paths to $\{a_1, b_1\}$ have length bounded by an absolute constant because a_1 and b_1 are just a (constant-sized) function of the inputs (c, x_1, y_1). If L is sufficiently large then this constant is at most L.
- For $j > 1$, paths from $\{c, x_1, y_1\}$ to a_j $(= a'_{j-1})$ consist of a path through a constant-sized circuit (the computation of c') concatenated with a path to a'_{j-1} in $\widetilde{\mathsf{Sort}}^{(2)}_{k-1,m-1}$. By induction the latter path has length at most $L \cdot (j-1)$, so if L is sufficiently large then the total path length is at most $L \cdot j$.
- For $i > 1$ and $j > 1$, paths from $\{x_i, y_i\}$ to $\{a_j, b_j\}$ are just paths from $\{x'_{i-1}, y'_{i-1}\}$ to $\{a'_{j-1}, b'_{j-1}\}$ and thus by induction they have length at most $L \cdot (j - i + 1)$. ◁

Returning to $\mathsf{Sort}^{(2)}_{k,m} = \widetilde{\mathsf{Sort}}^{(2)}_{k,m}(?, \cdot, \cdot)$ (without the tilde), we can see that any input-to-output path in $\mathsf{Sort}^{(2)}_{k,m}$ directly corresponds to an input-to-output path in $\widetilde{\mathsf{Sort}}^{(2)}_{k,m}$ whose length is appropriately bounded by Claim 11. That concludes the proof of Lemma 10. ◀

5 Unordered Multiselection Over Large Alphabets

In this section we construct a circuit that implements a relaxation of multiselection. Informally, this relaxation allows the output elements to appear in any order (and possibly with repetitions), rather than the same order in which they were queried. Additionally, rather than selecting individual bits (which is our eventual goal) - we consider a generalization to a larger alphabet size.

▶ **Definition 12.** Unordered multiselection *(with q queries, over alphabet Σ) is the search problem defined by the relation*

$$\widetilde{\mathsf{Sel}}^{n\to q}_{\Sigma} \stackrel{\mathsf{def}}{=} \left\{ \left((\mathbf{x}, \mathbf{i}), \mathbf{y}\right) \in \left(\Sigma^n \times [n]^q\right) \times (\Sigma \cup \{\perp\})^q \;:\; \left\{y_k : y_k \neq \perp\right\}_{k\in[q]} = \left\{x_{i_k}\right\}_{k\in[q]} \right\}.$$

5.1 Superconcentrators and Routing

We recall the notion of a superconcentrator, which is the main technical tool involved in our construction of Boolean circuits for unordered multiselection.

▶ **Definition 13** (Networks). *A* network $\mathcal{N}$ *is a tuple $\mathcal{N} = (V, E, A, B)$, where $G = (V, E)$ is a directed acyclic graph and $A, B \subseteq V$ are respectively sets of* sources *and* sinks *in G.*

If $|A| = |B| = n$, then $\mathcal{N}$ is said to be an n-network*. The* size *of $\mathcal{N}$, denoted by $|\mathcal{N}|$, is defined to be $|E|$; the* degree *of $\mathcal{N}$ is the degree of G; and the* depth *of $\mathcal{N}$ is the longest path length in G.*

A family of n-networks $\{\mathcal{N}_n\}_{n\in\mathbb{Z}^+}$ is said to be log-space uniform *if there an algorithm that on input 1^n runs in $O(\log n)$ space and outputs G (say with an adjacency list representation) and A, B (say represented by indicator strings).*

▶ **Definition 14** (Superconcentrators). *An n-network $\mathcal{N} = (V, E, A, B)$ is said to be a* superconcentrator *if for all $q \in [n]$ and all subsets $X \subseteq A$ and $Y \subseteq B$ with $|X| = |Y| = q$, there exist q vertex-disjoint paths from X to Y.*

▶ **Definition 15.** *Let $\mathcal{S} = (V, E, A, B)$ be an n-superconcentrator. The* routing problem *for $\mathcal{S}$ is a search problem $\mathsf{Route}_{\mathcal{S}}$ whose input is a pair of (indicator strings for) sets $X \subseteq A$, $Y \subseteq B$ with $|X| = |Y|$. A valid corresponding output is any (indicator string for a) set $R \subseteq E$ of edges such that R is a union of q vertex-disjoint paths from X to Y, where $q = |X| = |Y|$.*

Note that whenever $\mathcal{S}$ is an n-superconcentrator, the search problem $\mathsf{Route}_{\mathcal{S}}$ is total. Naturally, it is desirable for a superconcentrator $\mathcal{S}$ to admit efficient algorithms for $\mathsf{Route}_{\mathcal{S}}$. The state of the art in this respect is Pippenger's construction of a "self-routing" superconcentrator.

Informally, a delay-d self-routing protocol for a superconcentrator (V, E, A, B) associates each vertex $v \in V$ with a finite automaton that can communicate synchronously with the automata on neighboring vertices (where u is said to neighbor v if (u, v) or (v, u) is in E). For any sets $X \subseteq A$ and $Y \subseteq B$ with $|X| = |Y| = q$, if the automata in X and Y are initialized with state 1, and all other automata are initialized with state 0, then in d steps the automata jointly compute q vertex-disjoint paths from X to Y by on the d^{th} step transmitting 1 on all edges in those paths and 0 on other edges.

▶ **Definition 16.** *A* self-routing protocol *with* delay d *for an n-superconcentrator $\mathcal{S} = (V, E, A, B)$ is a tuple (Σ, δ), where:*

- *Σ is a finite "alphabet" set that contains $\{0, 1\}$; and*
- *$\delta : \Sigma^{V \cup E} \to \Sigma^{V \cup E}$ is a function such that:*
  - *for every vertex $v \in V$ with incident edges E_v, $\delta(\mathbf{q})_v$ depends as a function of $\mathbf{q}$ only on $(q_e)_{e \in E_v}$.*
  - *for every edge $e = (u, v) \in E$, $\delta(\mathbf{q})_e$ depends as a function of $\mathbf{q}$ only on q_u and q_v.*
- *For any sets $X \subseteq A$ and $Y \subseteq B$ with $|X| = |Y|$, if we define $\mathbf{q}^{(0)} \in \Sigma^{V \cup E}$ by*

$$q_v^{(0)} \stackrel{\mathsf{def}}{=} \begin{cases} 1 & \text{if } v \in X \cup Y \\ 0 & \text{if } v \in V \setminus (X \cup Y) \end{cases}$$

and $q_e^{(0)} \stackrel{\mathsf{def}}{=} 0$ for all $e \in E$, then the set

$$R \stackrel{\mathsf{def}}{=} \left\{ e \in E : \delta^d(\mathbf{q}^{(0)})_e = 1 \right\}$$

satisfies $\big((1_X, 1_Y), 1_R\big) \in \mathsf{Route}_{\mathcal{S}}$.

Such a self-routing protocol for an n-superconcentrator is said to be log-space uniform if there is a log-space Turing machine that on input 1^n outputs:

- *a description of Σ;*
- *for each $v \in V$ with incident edges E_v, the truth table for $\delta(\mathbf{q})_v$ as a function of $(q_e)_{e \in E_v}$.*
- *for each $e = (u, v) \in E$, the truth table for $\delta(\mathbf{q})_e$ as a function of (q_u, q_v).*

▶ **Definition 17.** *A* self-routing superconcentrator *is a superconcentrator $\mathcal{S}$ with an associated self-routing protocol (Σ, δ) for $\mathcal{S}$. A self-routing superconcentrator is said to be log-space uniform if both the superconcentrator and the associated self-routing protocol are log-space uniform.*

▶ **Theorem 18** ([25])**.** *There exists a log-space uniform self-routing n-superconcentrator $\mathcal{S}$ with size $O(n)$, degree $O(1)$, and depth $O(\log n)$, where the self-routing protocol has alphabet size $O(1)$ and delay $O(\log n)$.*

▶ **Corollary 19.** *There exists a log-space uniform n-superconcentrator $\mathcal{S}$ with size $O(n)$, degree $O(1)$, and depth $O(\log n)$, such that $\mathsf{Route}_{\mathcal{S}}$ is solvable by a log-space uniform Boolean circuit of size $O(n \log n)$ and depth $O(\log n)$.*

Proof. Let $\mathcal{S} = (V, E, A, B)$ be the log-space uniform self-routing superconcentrator given by Theorem 18, let the self-routing protocol be given by (Σ, δ), and let $d = O(\log n)$ denote the delay of this self-routing protocol. To prove the corollary, we need to construct a log-space uniform Boolean circuit for solving $\mathsf{Route}_{\mathcal{S}}$, with size $O(n \log n)$ and depth $O(\log n)$.

We are given 1_X and 1_Y. The circuit starts by computing $\mathbf{q}^{(0)} \in \Sigma^{V \cup E}$ as defined in Definition 16. By construction this can be done with a circuitry of size $O(n)$ and depth $O(1)$. The circuit next computes $\delta^d(\mathbf{q}^{(0)})$. Recall that each output symbol of $\delta(\mathbf{q})$ depends on at most $\deg(\mathcal{S}) = O(1)$ symbols of $\mathbf{q}$. Since the alphabet Σ is also constant-sized, this means that δ is implementable by a (log-space uniform) circuit of size $O(|V \cup E|) = O(n)$ and depth $O(1)$. Iterating this circuit d times, we compute $\delta^d(\mathbf{q}^{(0)})$ with log-space uniform circuitry of size $O(d \cdot n)$ and depth $O(d)$. Finally, we extract from $\delta^d(\mathbf{q}^{(0)})$ the indicator string 1_R for the set

$$R \stackrel{\text{def}}{=} \left\{e \in E : \delta^d(\mathbf{q}^{(0)})_e = 1\right\}.$$

This takes (log-space uniform) circuitry of size $O(|E|) = O(n)$ and depth $O(1)$.

In total, this circuit for computing 1_R is log-space uniform and has size $O(n \cdot d) = O(n \log n)$ and depth $O(d) = O(\log n)$. By the definition of a self-routing protocol, R satisfies $\left((1_X, 1_Y), 1_R\right) \in \mathsf{Route}_{\mathcal{S}}$, i.e. the circuit solves $\mathsf{Route}_{\mathcal{S}}$. ◀

5.2 From Superconcentrators To Unordered Multiselection

▶ **Proposition 20.** *Let $n \in \mathbb{Z}^+$, let $s = s(n)$ satisfy $s = \Omega(\log^2 n)$, and let $\Sigma = \{0,1\}^s$. For any $q = q(n) \in [n]$ there exists a log-space uniform Boolean circuit for $\widetilde{\mathsf{Sel}}_{\Sigma}^{n \to q}$ with size $O(n \cdot s)$ and depth $O(\log n)$.*

Proof. Let $\mathcal{S} = (V, E, A, B)$ be a log-space uniform n-superconcentrator with size $O(n)$, depth $O(\log n)$, degree $O(1)$ such that $\mathsf{Route}_{\mathcal{S}}$ is solvable by a log-space uniform Boolean circuit of size $O(n \log n)$ and depth $O(\log n)$. (Such an $\mathcal{S}$ is guaranteed to exist by Corollary 19). Order the elements of A and B arbitrarily so that $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$.

The Circuit. Our circuit, taking as input $\mathbf{x} \in \Sigma^n$ and $(i_1, \ldots, i_q) \in [n]^q$, is constructed in two parts.

The first part of our circuit computes indicator strings $1_X \in \{0,1\}^A$, $1_Y \in \{0,1\}^B$, and $1_R \in \{0,1\}^E$ for sets $X \subseteq A$, $Y \subseteq B$, and $R \subseteq E$ defined as follows. The set X is defined as $\{a_{i_1}, \ldots, a_{i_q}\}$, Y is defined as $\{b_1, \ldots, b_{q'}\}$, where q' is the cardinality of X (which may be smaller than q due to repetitions), and R is the edges of q' vertex-disjoint paths from X to Y.

The second part of our circuit consists of a gadget g_v for each vertex $v \in V$. For a vertex $v \in A$, say $v = a_j$, the gadget g_v simply outputs x_j if $v \in X$ and $\perp$ otherwise. For other v, i.e. with positive in-degree d, the gadget g_v is a circuit for $\mathsf{Sel}_{\Sigma \cup \{\perp\}}^{d \to 1}$ (e.g. the circuit from Proposition 2). To construct the inputs for g_v, first order the in-neighbors of v arbitrarily as $u_1, \ldots, u_d$. The data input for g_v is then constructed by taking the i^{th} symbol, for any $i \in [d]$, to be the output of g_{u_i}. The selector input for g_v is constructed to be (the unique) $i^\star \in [d]$ such that $(u_{i^\star}, v) \in R$ if such an $i^\star$ exists, and arbitrary otherwise.

Finally, the outputs of our circuit are the outputs of $g_{b_1}, \ldots, g_{b_q}$.

Size and Depth. In the first part of our circuit, the computation of 1_X uses log-space uniform circuitry of size $O(n \log^2 n) = O(n \cdot s)$ and depth $O(\log n)$ by Lemma 21 below. Next, 1_Y is obtained by sorting 1_X with log-space uniform circuitry of size $O(n \log n)$ and depth $O(\log n)$ (this is possible by Lemma 8). Finally, the computation of 1_R uses log-space uniform circuitry of size $O(n \log n)$ and depth $O(\log n)$ because of our choice of superconcentrator $\mathcal{S}$. In total the circuitry of this part has size $O(n \cdot s)$ and depth $O(\log n)$.

In the second part of our circuit, there are $O(n)$ gadgets (because $\mathcal{S}$ has size $O(n)$), and each gadget has size $O(s)$ (because $\mathcal{S}$ has constant degree). Additionally for each non-source gadget there is constant-sized circuitry to compute the selector input as a function of 1_R (again because $\mathcal{S}$ has constant degree). In total the circuitry size is $O(n \cdot s)$. As for depth, each gadget has constant depth. Since the output of gadget u is an input to gadget v iff (u, v) is an element of E, and $\mathcal{S}$ has depth $O(\log n)$, the circuitry here also has depth $O(\log n)$.

Correctness. The main claim that we use to establish correctness is that for every $a_i \in X$ and every vertex $v \in V$, if v lies on a path in R from a_i to B, then the output of g_v is x_i, and otherwise the output of g_v is $\perp$. Here a_i is well-defined as a partial function of v because R consists only of vertex-disjoint paths. This claim suffices for correctness because there are paths in R from each a_{i_k} to B, but not from any $a \in A \setminus \{i_1, \ldots, i_q\}$, so the set of non-$\perp$ outputs is exactly $\{x_{i_1}, \ldots, x_{i_q}\}$.

We prove the claim by induction on the depth of v (i.e. the maximum length of a path ending in v). The base case is when v has depth 0, i.e. v is a source vertex a_i in (V, E). In this case we *defined* g_v to output x_i.

For the inductive step, consider a vertex v with positive depth, and suppose that the claim holds for all u of lesser depth, and in particular for the in-neighbors $u_1, \ldots, u_d$ of v. Now if v lies on a path p from some a_i to Y, this path must go through u_j for some $j \in [d]$, and then the inductive hypothesis implies that the output of g_{u_j} is x_i. The construction of the selector input for g_v ensures that g_v also outputs x_i, which completes the proof of the claim. ◀

5.3 Computing Set Indicator Strings

▶ **Lemma 21.** *For $q = q(n) \in \mathbb{Z}^+$, there is a log-space uniform Boolean circuit with size $O\big((n+q) \cdot \log^2(n+q)\big)$ and depth $O\big(\log(n+q)\big)$ that maps $(i_1, \ldots, i_q) \in [n]^q$ to the indicator string $1_{\{i_1,\ldots,i_q\}} \in \{0,1\}^n$.*

Proof. On input $(i_1, \ldots, i_q)$, the circuit performs the following steps:

1. Construct the list $(1, \ldots, n, i_1, \ldots, i_q) \in [n]^{n+q}$, and sort it to obtain a non-decreasing list $(j_i)_{i \in [n+q]}$ with the property that j_i appears more than once in the list iff $j_i \in \{i_1, \ldots, i_q\}$. This can be done with circuitry of size $O\big((n+q) \cdot \log^2(n+q)\big)$ and depth $O\big(\log(n+q)\big)$ by Lemma 6.
2. Label j_i with 0 if the value j_i appears once in the list. If j_i appears more than once, we label the first occurrence with 1 and all other occurrences with $\perp$. This labels j_i with 1 only if j_i is in $\{i_1, \ldots, i_q\}$, and labels j_i with 0 only if it isn't.
   More explicitly, we compute labels
   $$b_i = \begin{cases} \perp & \text{if } i > 1 \text{ and } j_{i-1} = j_i \\ 1 & \text{otherwise, if } i < n+q \text{ and } j_i = j_{i+1} \\ 0 & \text{otherwise} \end{cases}$$
   for $i \in [n+q]$. This can be done with circuitry of size $O\big((n+q)\log n\big)$ and depth $O(\log \log n)$: first compute in parallel for each $i \in [n+q-1]$ whether or not $j_i = j_{i+1}$ (this circuit has size $O(\log n)$ and depth $O(\log \log n)$), and then compute each b_i with a constant-sized circuit.
3. Sort the list $\big((j_i, b_i)\big)_{i \in [n+q]}$ in order of increasing j_i, except that if $b_i = \perp$ we treat j_i as $+\infty$. That is, we prepend each j_i with 1 if $b_i = \perp$ and with 0 otherwise. We then take the first n elements of the result to obtain a list $\big((i, b'_i)\big)_{i \in [n]}$ such that $b'_i \in \{0,1\}$ satisfies

$$b'_i = \begin{cases} 1 & \text{if } i \in \{i_1, \ldots, i_q\} \\ 0 & \text{otherwise.} \end{cases}$$

In other words, $(b'_1, \ldots, b'_n)$ is $1_{\{i_1,\ldots,i_q\}}$, which is our desired output. This step can be done with circuitry of size $O\big((n+q)\cdot \log^2(n+q)\big)$ and depth $O\big(\log(n+q)\big)$ by Lemma 6. ◀

6 From Unordered to Ordered Multiselection

In this section we construct a circuit for *ordered* multiselection over a large alphabet from a circuit for *unordered* multiselection over a slightly larger alphabet.

▶ **Lemma 22.** *Let $q = q(n) \in [n]$ and $s = s(n) = \Omega(\log n)$ be integer functions of n, let $\Sigma = \{0,1\}^s$, let $\widetilde{\Sigma} = [n] \times \Sigma$, and suppose there is a (log-space uniform) Boolean circuit for $\widetilde{\mathsf{Sel}}^{n/s \to q}_{\widetilde{\Sigma}}$ with size S and depth D.*

Then there is a (log-space uniform) Boolean circuit for $\mathsf{Sel}^{n\to q}_{\Sigma}$ with size $S + O(q \cdot s \cdot \log q)$ and depth $D + O(\log n)$.

Combining Lemma 22 with Proposition 20 gives the following corollary.

▶ **Corollary 23.** *Let $s = s(n)$ satisfy $s = \Omega(\log^2 n)$, let $\Sigma = \{0,1\}^s$, and let $q = q(n) \in [n]$. Then there is a log-space uniform Boolean circuit for $\mathsf{Sel}^{n\to q}_{\Sigma}$ with size $O(n \cdot s + q \cdot s \cdot \log n)$ and depth $O(\log n)$.*

6.1 Boolean Circuits for Inner Joins

The main tool that we use in the proof of Lemma 22 is Boolean circuitry for computing an *inner join* of two relations (cf. relational databases [12]).

▶ **Definition 24.** *If $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$ and $\mathcal{S} \subseteq \mathcal{X} \times \mathcal{Y}$ are relations, the* inner join *of $\mathcal{R}$ and $\mathcal{S}$ (which we denote by $\mathcal{R} \bowtie \mathcal{S}$) is*

$$\mathcal{R} \bowtie \mathcal{S} \stackrel{\text{def}}{=} \big\{(w,x,y) : (w,x) \in \mathcal{R} \ \wedge \ (x,y) \in \mathcal{S}\big\}.$$

To our knowledge, prior work on the computational complexity of computing joins has focused on algorithms in the RAM or PRAM models of computation, as opposed to Boolean circuits.

We focus for simplicity on a special case. First, we require that the relation $\mathcal{S}$ is a partial function. That is, for every $x \in \mathcal{X}$ there is at most one $y \in \mathcal{Y}$ such that $(x,y) \in \mathcal{S}$. The partial function requirement prevents $|\mathcal{R} \bowtie \mathcal{S}|$ from being larger than $|\mathcal{R}|$. We also require that for every $(w,x) \in \mathcal{R}$ there exists some $(x,y) \in \mathcal{S}$.

The following proposition says that inner joins in this special case are computable by (uniform) Boolean circuits of logarithmic depth and nearly linear size.

▶ **Proposition 25.** *Let n, m be positive integers, let $\mathcal{W}$ and $\mathcal{Y}$ be sets whose elements are represented by s-bit strings, and let $\mathcal{X}$ be a finite set whose elements are represented by k-bit strings.*

There exists a log-space uniform Boolean circuit of size $O(q \cdot (k+s) \cdot \log q)$ and depth $O(k + \log q)$ that takes as input a relation $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$ and a partial function $f \subseteq \mathcal{X} \times \mathcal{Y}$ with $|\mathcal{R}| \le q$, $|f| \le q$, and $\{x : \exists w, (w,x) \in \mathcal{R}\} \subseteq \{x : \exists y, (x,y) \in f\}$, and outputs $\mathcal{R} \bowtie f$.

Here sets are represented by an arbitrarily ordered listing of their elements, padded with $\perp$ elements as needed to have length q.

Proof. Denote the elements of $\{x : \exists y, (x,y) \in f\}$ by $x_1, \ldots, x_k$, where $x_1 < \cdots < x_k$. For $i \in [k]$, let W_i denote the set $\{w : (w, x_i) \in \mathcal{R}\}$, and let n_i denote $|W_i|$. With this notation, our desired output is a list of length q whose non-$\perp$ elements in some order are precisely $\big(w, x_i, f(x_i)\big)_{i\in[k], w\in W_i}$.

We first construct a list with at most $2q$ elements in $(\mathcal{W} \cup \{\star\}) \times \mathcal{X} \times (\mathcal{Y} \cup \{\star\})$, namely $\big\{(x_i, w, \star)\big\}_{i\in[k], w\in W_i}$ and $\Big\{\big(x_i, \star, f(x_i)\big)\Big\}_{i\in[k]}$. Such a list is readily obtained by concatenating the listings of $\mathcal{R}$ and f (with the natural embeddings of $\mathcal{W} \times \mathcal{X}$ and $\mathcal{X} \times \mathcal{Y}$ in $(\mathcal{W} \cup \star) \times \mathcal{X} \times (\mathcal{Y} \cup \star)$).

We sort this list with respect to the partial ordering that defines $(w, x, y) \prec (w', x', y')$ iff $x < x'$ or $x = x'$ and $w = \star$ and $w' \neq \star$. By Lemma 6 this can be done with circuitry of size $O(q \cdot (k + s) \cdot \log q)$ and depth $O(\log q)$. This yields a list $\mathcal{L}$ composed of k blocks, the i^{th} of which has length $n_i + 1$ and is

$$\Big(\big(\star, x_i, f(x_i)\big), (w_{i,1}, x_i, \star), \ldots, (w_{i,n_i}, x_i, \star)\Big),$$

where $\{w_{i_1}, \ldots, w_{i,n_i}\} = W_i$.

With local processing, we obtain two lists $\mathcal{L}_f$ and $\mathcal{L}_\mathcal{S}$ where the i^{th} block of $\mathcal{L}_f$ is

$$\big(f(x_i), \underbrace{\perp, \ldots, \perp}_{n_i \text{ times}}\big) \tag{4}$$

and the i^{th} block of $\mathcal{L}_\mathcal{S}$ is

$$\big(\perp, (w_{i,1}, x_i), \ldots, (w_{i,n_i}, x_i)\big). \tag{5}$$

Using Lemma 26 below, we map $\mathcal{L}_f$ to a list whose i^{th} block is

$$\big(\underbrace{f(x_i), \ldots, f(x_i)}_{n_i + 1 \text{ times}}\big). \tag{6}$$

We then locally combine (6) with (5) to obtain a list whose i^{th} block is

$$\Big(\perp, \big((w_{i,1}, x_i, f(x_i)\big), \ldots, (w_{i,n_i}, x_i, f(x_i))\big)\Big). \tag{7}$$

A final sorting step sends the $\perp$ elements to the back of the list, which allows us to conclude by truncating the list to length n. This step can also be done with circuitry of size $O(q \cdot (k + s) \cdot \log q)$ and depth $O(\log q)$ by Lemma 6. ◀

▶ **Lemma 26.** *For $n, s \in \mathbb{Z}^+$, there is a (logspace-uniform) constant fan-in Boolean circuit of size $O(n \cdot s)$ and depth $O(\log n)$ that takes as input $\mathbf{x} \in (\{0,1\}^s \cup \{\perp\})^n$ with $x_1 \neq \perp$, and outputs $\mathbf{y} \in \big(\{0,1\}^s\big)^n$ such that*

$$y_i = \begin{cases} x_i & \text{if } x_i \neq \perp \\ y_{i-1} & \text{otherwise.} \end{cases}$$

Equivalently, $y_i = x_{j_i}$, where $j_i = \max\{j : 1 \leq j \leq i \wedge x_j \neq \perp\}$ (this maximum is guaranteed to be over a non-empty set because $x_1 \neq \perp$).

Proof. Without loss of generality we can assume that $s = 1$ because for $s > 1$ we can use s copies of the circuit for the $s = 1$ case.

Consider the binary operation $\star : \{0,1,\perp\} \times \{0,1,\perp\} \to \{0,1,\perp\}$ defined by

$$\tau \star \upsilon = \begin{cases} \tau & \text{if } \upsilon = \perp \\ \upsilon & \text{if } \upsilon \neq \perp. \end{cases}$$

It is clear that our desired $\mathbf{y}$ has $y_1 = x_1$ and $y_i = y_{i-1} \star x_i$ for $i > 1$.

We now observe that $\star$ is an associative operation. To see this, consider any $\sigma, \tau, \upsilon \in \{0,1,\perp\}$ and consider separately the cases $\upsilon = \perp$ and $\upsilon \neq \perp$. If $\upsilon = \perp$ then $(\sigma \star \tau) \star \upsilon = \sigma \star \tau = \sigma \star (\tau \star \upsilon)$. If $\upsilon \neq \perp$ then $(\sigma \star \tau) \star \upsilon = \upsilon = \sigma \star \upsilon = \sigma \star (\tau \star \upsilon)$.

Now we use a classic result of Ladner and Fischer [20] that for *any* associative operation $\star : \Sigma \times \Sigma \to \Sigma$ and $n \in \mathbb{Z}^+$, there exists a (log-space uniform) circuit of size $O(n)$ and depth $O(\log n)$ (with only $\star$-gates) that computes all $\star$-prefix products. That is, the circuit takes as input $\mathbf{x} \in \Sigma^n$ and outputs $\mathbf{y}$ such that $y_i = x_1 \star \cdots \star x_i$.

In our case, $\star$ is computable by a Boolean circuit of constant size, so replacing $\star$-gates by such a circuit finishes the proof. ◀

6.2 A Circuit for Ordered Multiselection

We are now ready to prove Lemma 22.

Proof of Lemma 22. Let $\tilde{C}$ be a circuit for $\widetilde{\mathsf{Sel}}_{\widetilde{\Sigma}}^{n \to q}$.

The Circuit. Our circuit for $\mathsf{Sel}_{\Sigma}^{n \to q}$ takes as input $(\mathbf{x}, \mathbf{i}) \in \Sigma^n \times [n]^q$ and consists of the following stages:

1. Compute $\tilde{\mathbf{x}} \in \widetilde{\Sigma}^n$, defined so that $\tilde{x}_i = (i, x_i)$ for $i \in [n]$, and compute a list representation of

   $$\mathcal{I} = \{(1, i_1), \ldots, (q, i_q)\}.$$

2. Compute $\tilde{\mathbf{y}} \leftarrow \tilde{C}(\tilde{\mathbf{x}}, \mathbf{i})$ to obtain $\tilde{\mathbf{y}} \in \left(\tilde{\Sigma} \cup \{\perp\}\right)^q$ such that

   $$\{\tilde{y}_k : \tilde{y}_k \neq \perp\}_{k \in [q]} = \{\tilde{x}_{i_k}\}_{k \in [q]} = \left\{(i_k, x_{i_k})\right\}_{k \in [q]}.$$

   In particular, each $\tilde{y}_k$ that is not equal to $\perp$ has the form (i, x_i) for some $i \in [n]$.
3. Deduplicate $\tilde{\mathbf{y}}$ to obtain a list representation of the partial function

   $$\mathcal{Y} = \left\{(i_k, x_{i_k})\right\}_{k \in [q]}.$$

   This involves sorting the non-$\perp$ symbols $\left\{(i_k, x_{i_k})\right\}$ of $\tilde{\mathbf{y}}$ in order of increasing i_k (this can be done with circuitry of size $O(n \cdot s \cdot \log n)$ and depth $O(\log n)$ by Lemma 6).
4. Apply the circuit given by Proposition 25 to $\mathcal{I}$ and $\mathcal{Y}$ to compute a list representation of

   $$\mathcal{I} \bowtie \mathcal{Y} = \left\{(k, i_k, x_{i_k})\right\}_{k \in [q]}.$$

5. Sort the list representation of $\mathcal{I} \bowtie \mathcal{Y}$ in order of increasing k, and read off the desired output $(x_{i_1}, \ldots, x_{i_k})$.

Size and Depth. The "computation" of $\tilde{\mathbf{x}}$ from $\mathbf{x}$ and of $\mathcal{I}$ from $\mathbf{i}$ is just adding constant values, so it can certainly be done by a (log-space uniform) Boolean circuit of size $O\big(n \cdot (s + \log n)\big)$ and depth $O(1)$.

By assumption, $\tilde{C}$ is a circuit of size S and depth D.

Via sorting (Lemma 6), one can deduplicate $\tilde{\mathbf{y}}$ with circuitry of size $O(q \cdot (s + \log n) \cdot \log q) = O(q \cdot s \cdot \log q)$ and depth $O(\log n)$.

By Proposition 25, the computation of $\mathcal{I} \bowtie \mathcal{Y}$ can also be done with circuitry of size $O(q \cdot (s + \log n) \cdot \log q) = O(q \cdot s \cdot \log n)$ and depth $O(\log n)$.

Sorting the elements of $\mathcal{I} \bowtie \mathcal{Y}$ again takes log-space uniform circuitry of size $O(q \cdot s \cdot \log q)$ and depth $O(\log n)$ by Lemma 6.

In total our constructed circuit has size $S + O(q \cdot s \cdot \log q)$ and depth $D + O(\log n)$. ◀

7 Binary Multiselection and Proof of Main Theorem

Next, we use the multiselection circuit over large alphabets to obtain a *binary* multiselection circuit.

▶ **Lemma 27.** *Let $q = q(n)$ and $s = s(n)$ be integer functions of n, let $\Sigma = \{0,1\}^s$, and let C_n be a (log-space uniform) Boolean circuit for $\mathsf{Sel}_{\Sigma}^{n/s \to q}$ with size S and depth D.*

Then there is a (log-space uniform) Boolean circuit C'_n for $\mathsf{Sel}_{\{0,1\}}^{n \to q}$ with size $S + O(q \cdot s)$ and depth $D + O(\log s)$.

Proof. On data input $\mathbf{x} \in \{0,1\}^n$ and selector input $(i_1, \ldots, i_q) \in [n]^q$, C'_n performs the following steps:

1. View $\mathbf{x} \in \{0,1\}^n$ as $\mathbf{X} \in \Sigma^{n/s}$ by setting $X_i = \big(x_{(i-1)\cdot s+1}, \ldots, x_{i \cdot s}\big)$. View each i_j as a $\log n$-bit string with prefix $p_j \in \{0,1\}^{\log n - \log s}$ and a suffix $s_j \in \{0,1\}^{\log s}$. This is just relabeling wires, so it requires no circuitry.
2. Compute $(X_{p_1}, \ldots, X_{p_q}) \leftarrow C_n\big(\mathbf{X}, (p_1, \ldots, p_q)\big)$. This requires circuitry of size S and depth D.
3. For each $j \in [q]$ in parallel, compute $x_{i_j} = \mathsf{Sel}_{\{0,1\}}^{s \to 1}\big(X_{p_j}, (s_j)\big)$. By Proposition 2, for each $j \in [q]$ this requires circuitry of size $O(s)$ and depth $O(\log s)$, for a total circuitry size of $O(q \cdot s)$ and depth $O(\log s)$.
4. Output $(x_{i_1}, \ldots, x_{i_q})$. ◀

We can now prove our main theorem, which we recall here for convenience.

[Main Theorem] For all $n, q \in \mathbb{Z}^+$ there is a Boolean circuit computing $\mathsf{Sel}^{n \to q}$ of size $O(n + q \cdot \log^3(n))$ and depth $O(\log(n + q))$.

Proof. Set $s(n) = \log^2 n$ and let $\Sigma = \{0,1\}^s$. By Corollary 23, there exists a log-space uniform circuit for $\mathsf{Sel}_{\Sigma}^{n/s \to q}$ with size $O(s \cdot n/s + q \cdot s \cdot \log n) = O(n + q \log^3 n)$ and depth $O\big(\log(n/s)\big) = O(\log n)$. Applying Lemma 27 to this circuit yields a log-space uniform circuit for $\mathsf{Sel}_{\{0,1\}}^{n \to q}$ with size $O\big(n + q \cdot (\log^3 n + s)\big) = O(n + q \cdot \log^3 n)$ and depth $O(\log n + \log s) = O(\log n)$. ◀

References

1 M. Ajtai, J. Komlós, and E. Szemerédi. An 0(n log n) sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 1–9, New York, NY, USA, 1983. Association for Computing Machinery. doi:10.1145/800061.808726.

2 Alexander Andreev. On a method for obtaining more than quadratic effective lower bounds for the complexity of π-scheme. *Moscow University Mathematics Bulletin*, 42(1):63–66, 1987.

3 Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. Optorama: Optimal oblivious RAM. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 403–432. Springer, 2020. doi:10.1007/978-3-030-45724-2_14.

4 Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Enoch Peserico, and Elaine Shi. Oblivious parallel tight compaction. In *ITC*, volume 163 of *LIPIcs*, pages 11:1–11:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

5 Gilad Asharov, Wei-Kai Lin, and Elaine Shi. Sorting short keys in circuits of size $o(n \log n)$. *SIAM J. Comput.*, 51(3):424–466, 2022. doi:10.1137/20m1380983.

6 Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2000.

7 Norbert Blum. A boolean function requiring 3n network size. *Theor. Comput. Sci.*, 28:337–345, 1984. doi:10.1016/0304-3975(83)90029-4.

8 Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In *TCC (2)*, volume 10678 of *Lecture Notes in Computer Science*, pages 662–693. Springer, 2017.

9 Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In *TCC (2)*, volume 11892 of *Lecture Notes in Computer Science*, pages 407–437. Springer, 2019.

10 Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In *TCC (2)*, volume 10678 of *Lecture Notes in Computer Science*, pages 694–726. Springer, 2017.

11 Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998. doi:10.1145/293347.293350.

12 Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

13 Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.

14 Justin Holmgren and Ron D. Rothblum. Faster sounder succinct arguments and IOPs. In *CRYPTO (1)*, volume 13507 of *Lecture Notes in Computer Science*, pages 474–503. Springer, 2022.

15 Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *STOC*, pages 262–271. ACM, 2004.

16 Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732. ACM, 1992.

17 É Sh Kospanov. Scheme realization of the sorting problem. *Diskretnyi Analiz i Issledovanie Operatsii*, 1(1):13–19, 1994.

18 Michal Koucký and Karel Král. Sorting short integers. In *ICALP*, volume 198 of *LIPIcs*, pages 88:1–88:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

19 Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 364–373. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646125.

20 Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *J. ACM*, 27(4):831–838, October 1980. doi:10.1145/322217.322232.

21 Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic ram computation from ring lwe. Cryptology ePrint Archive, Paper 2022/1703, 2022. URL: https://eprint.iacr.org/2022/1703.

22 Wei-Kai Lin and Elaine Shi. Optimal sorting circuits for short keys. *CoRR*, abs/2102.11489, 2021. arXiv:2102.11489.

23 È. I. Nechiporuk. A Boolean function. *Sov. Math., Dokl.*, 7:999–1000, 1966.

24 Wolfgang J. Paul. A 2.5 n-lower bound on the combinational complexity of boolean functions. *SIAM J. Comput.*, 6(3):427–443, 1977. doi:10.1137/0206030.

25 Nicholas Pippenger. Self-routing superconcentrators. *J. Comput. Syst. Sci.*, 52(1):53–60, 1996. doi:10.1006/jcss.1996.0005.

26 Noga Ron-Zewi and Ron D. Rothblum. Proving as fast as computing: succinct arguments with constant prover overhead. In *STOC*, pages 1353–1363. ACM, 2022.

27 John E. Savage. *Models of computation - exploring the power of computing.* Addison-Wesley, 1998.

28 Leslie G. Valiant. On non-linear lower bounds in computational complexity. In William C. Rounds, Nancy Martin, Jack W. Carlyle, and Michael A. Harrison, editors, *Proceedings of the 7th Annual ACM Symposium on Theory of Computing, May 5-7, 1975, Albuquerque, New Mexico, USA*, pages 45–53. ACM, 1975. doi:10.1145/800116.803752.

A Applications

While we find the multiselection problem to be basic and natural, we additionally mention two concrete applications of the linear-sized multiselection circuit of Section 1 to problems in cryptography.

A.1 Application 1: Simplifying Efficient Arguments

Recently there has been a great deal of interest, both in theory and in practice, in developing efficient argument-systems (aka computationally sound proofs), proving for example that a given formula is satisfiable with a proof that is much shorter than the satisfying assignment (and is also very efficiently verifiable). A key bottleneck in such proof-systems is the efficiency of *proving* correctness, relative to the cost of merely computing the function.

Recent works [26, 14] consider the setting of Boolean circuits and construct provers whose size is *linear* in the size of the original computation. These works, following Kilian's pioneering work [16], use a (generalization of a) PCP which is compiled into a succinct argument by sending a short hash of the PCP and then decommiting to desired locations that are sampled by the verifier. This naturally requires multiselection: the prover needs to restrict the PCP proof string to the selected indices. Thus, to get a linear-size prover, these works need a linear-size multiselection gadget.

Both [26] and [14] relied on ad-hoc solutions leveraging application-specific structure of the queries $(i_1, \ldots, i_q)$ and expended considerable effort to guarantee this structure. For example in the case of [14], a new local testing procedure was presented for tensor codes, with the novel property that the local testing queries were efficiently "multiselectable".

Section 1 makes this work unnecessary by removing the burden of worrying about a particular query structure. This significantly simplifies these works (especially [14]) and is likely to simplify similar future works.

A.2 Application 2: More Efficient Batch PIR

Private information retrieval (PIR) [11, 19] is a process by which a client with an index $i \in [n]$ obtains an element x_i from a server with a database $\mathbf{x} \in \Sigma^n$ while (computationally) hiding all information about i from the server. *Batch PIR* has just one modification: the client has multiple indices $i_1, \ldots, i_q$, and correspondingly obtains $x_{i_1}, \ldots, x_{i_q}$. We call q the *batch size*. Thus the standard notion of PIR, which we also refer to as *non-batch* PIR, corresponds to the case $q = 1$. The raison d'être of batch PIR is that the server's running time can be much less than q times the cost of non-batch PIR.

Indeed, our Section 1 implies the following corollary: if the ring learning with errors[2] (ring LWE) assumption holds, then for every batch size $q \leq n/\log^3 n$ and every constant $\epsilon > 0$, there is a batch PIR protocol in which:

- the server's running time is $n \cdot \text{polylog}(\lambda)$, where λ is a computational security parameter,
- the client-to-server communication is $(1+\epsilon) \cdot q \cdot \log n + \text{poly}(\lambda)$,
- the server-to-client communication is $(1+\epsilon) \cdot q + \text{poly}(\lambda)$, and
- the client's running time is $q \cdot \log n \cdot \text{polylog}(\lambda) + \text{poly}(\lambda)$.

In a nutshell, the idea is for the client to send an encryption $\mathsf{ct}_{\mathbf{i}}$ of the indices $\mathbf{i} = (i_1, \ldots, i_q)$ under a fully homomorphic encryption (FHE) scheme with the appropriate efficiency properties. The server, holding database $x \in \{0,1\}^n$, homomorphically evaluates $\mathsf{Sel}^{n \to q}(x, \cdot)$ (represented by the circuit of Section 1) on $\mathsf{ct}_{\mathbf{i}}$ and sends the result to the client, which decrypts to obtain its output.

As for efficiency, homomorphic evaluation should satisfy two properties. First, the cost of homomorphically evaluating a circuit C should be $|C| \cdot \text{polylog}(\lambda)$ (at least when C is the circuit for $\mathsf{Sel}^{n \to q}$ constructed in Section 1). Second, the ciphertexts resulting from homomorphic evaluation should have rate 1. If the ring LWE assumption holds, then one way to obtain such an FHE scheme is by combining the work of [13], which constructs FHE with $\text{polylog}(\lambda)$ evaluation overhead[3], with the work of [9], which for any constant $\epsilon > 0$ constructs FHE in which evaluated ciphertexts have rate $1 - \epsilon$.

A.3 Prior Work

A.3.1 Batch PIR via Batch Codes

One major alternative approach to batch PIR is a reduction to non-batch PIR using *batch codes* [15], which encode a database $\mathbf{x} \in \{0,1\}^n$ as a string $\mathbf{X} \in \left(\{0,1\}^{N/m}\right)^m$ such that to recover any q bits of $\mathbf{x}$, it suffices to read one bit from each N/m-bit block of $\mathbf{X}$. Here N, m, and q are parameters of the batch code. By retrieving each of these m bits with a non-batch PIR protocol, we obtain a batch PIR protocol with batch size q, server running time $\approx N$, and communication $\approx m$.

To replicate our batch PIR result via this approach, one would need batch codes with $m \approx k$ and $N \approx n$ for q at least $\omega(1)$. However, no such codes are known (see the table in Section 1.2 of [15]).

A.3.2 Doubly Efficient PIR.

A recent breakthrough result of Lin, Mook, and Wichs [21] shows how to achieve *doubly efficient PIR (DEPIR)* [6, 10, 8], i.e. PIR where the server's running time is sublinear in the database length (after a one-time deterministic preprocessing step), under the ring LWE assumption. Among other benefits, this allows the server's work to increase sublinearly with the number of queries, similarly to batch PIR. Amazingly, unlike in batch PIR, this is true even if the queries come from independent clients.

One can generically construct batch PIR protocols from DEPIR protocols, although as we now explain, our construction of batch PIR is more efficient than what one would obtain from [21]. Their DEPIR protocol exhibits a tunable trade-off between the server's online time and preprocessing time. They present two specific parameter settings:

[2] If we instead assume only standard LWE, we obtain a similar result but with all $\text{polylog}(\lambda)$ factors replaced by $\text{poly}(\lambda)$.

[3] The work of [13] obtains this overhead only for circuits of width at least λ.

- **(Online-Optimized)** The online time is $\text{polylog}(n) \cdot \text{poly}(\lambda)$, but the preprocessing time is $n^{1+\epsilon} \cdot \text{poly}(\lambda)$ for any constant $\epsilon > 0$.
- **(Preprocessing-Optimized)** The preprocessing time is $n \cdot 2^{\tilde{\Theta}(\sqrt{\log n})} \cdot \text{poly}(\lambda)$, but the online time is $2^{\tilde{\Theta}(\sqrt{\log n})} \cdot \text{poly}(\lambda)$.

In contrast to our batch PIR construction, their doubly efficient PIR server's *total* running time with either parameter setting never depends only linearly on n, and the server's per-query running time (including the amortized cost of pre-processing) is not polylogarithmic in n unless the number of queries is larger than the database.

B Linear-time Uniformity

In this section we briefly discuss an extension of our multiselection circuit that can be generated by a *linear-time* algorithm (rather than log-space uniformity as in Section 1). The specific notion of uniformity that we consider here is constructability by an algorithm in the standard word RAM model, that on input 1^n, runs in $O(n)$ time, using words of size $O(\log n)$ and a standard instruction set.

▶ **Theorem 28** (Linear-time Uniformity). *There exists a constant $\epsilon > 0$ such that for every $q = q(n) \leq n^{\epsilon}$, there exists a* linear-time uniform *Boolean circuit computing $\mathsf{Sel}^{n \to q}$ of linear-size and logarithmic-depth.*

Proof Sketch. As a matter of fact, we show how one can generically take any n^c-time-uniform multiselection circuit for $c \geq 1$ (e.g., those of Section 1) and transform it into a *linear-time uniform* multiselection circuit (for a somewhat smaller number of queries), while preserving the linear-size and logarithmic depth.

The idea is to first generate a multiselection circuit C for input strings of length $n' = n^{1/c}$. By assumption, this can be done in time $O\big((n')^c\big) = O(n)$. The multiselection is then constructed as follows: we partition the input string $x \in \{0,1\}^n$ into n' blocks of size $n/n' = n^{1-1/c}$ bits each. We then run $n^{1-1/c}$ copies of C, where the i-th copy is given the i-th bit of each of the blocks. The output of these circuit copies of C can be interpreted as q blocks in which our desired indices lie. We then apply a direct (uni-)selector to each block to obtain the desired bit. This step can be implemented by q circuits, each of size $O(n^{1-1/c})$. As long as $q = O(n^{1/c})$, the constructed circuit has linear-size and logarithmic depth.

To generate the circuit, our algorithm first generates the base multi-selector circuit which as noted above, takes time $O(n)$. Once that circuit is generated, creating the $n^{1-1/c}$ copies (with suitable input wiring) can be done in time $O(n^1/c \cdot n^{1-1/c}) = O(n)$. The additional circuitry can also be constructed in $O(q \cdot n^{1-1/c}) = O(n)$ time. ◀