Complexity of Linear Operators

Alexander S. Kulikov
Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences,
St. Petersburg State University, Russia
https://logic.pdmi.ras.ru/~kulikov/
kulikov@logic.pdmi.ras.ru

Ivan Mikhailin
University of California, San Diego, CA, USA
imikhail@eng.ucsd.edu

Andrey Mokhov
School of Engineering, Newcastle University, UK
andrey.mokhov@ncl.ac.uk

Vladimir Podolskii
Steklov Mathematical Institute, Russian Academy of Sciences, Moscow, Russia
http://www.mi-ras.ru/~podolskii/
podolskii@mi-ras.ru

Abstract

Let $A \in \{0,1\}^{n\times n}$ be a matrix with z zeroes and u ones and x be an n-dimensional vector of formal variables over a semigroup $(S, \circ)$. How many semigroup operations are required to compute the linear operator Ax?

As we observe in this paper, this problem contains as a special case the well-known range queries problem and has a rich variety of applications in such areas as graph algorithms, functional programming, circuit complexity, and others. It is easy to compute Ax using $O(u)$ semigroup operations. The main question studied in this paper is: can Ax be computed using $O(z)$ semigroup operations? We prove that in general this is not possible: there exists a matrix $A \in \{0,1\}^{n\times n}$ with exactly two zeroes in every row (hence $z = 2n$) whose complexity is $\Theta(n\alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function. However, for the case when the semigroup is commutative, we give a constructive proof of an $O(z)$ upper bound. This implies that in commutative settings, complements of sparse matrices can be processed as efficiently as sparse matrices (though the corresponding algorithms are more involved). Note that this covers the cases of Boolean and tropical semirings that have numerous applications, e.g., in graph theory.

As a simple application of the presented linear-size construction, we show how to multiply two $n \times n$ matrices over an arbitrary semiring in $O(n^2)$ time if one of these matrices is a 0/1-matrix with $O(n)$ zeroes (i.e., a complement of a sparse matrix).

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases algorithms, linear operators, commutativity, range queries, circuit complexity, lower bounds, upper bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.17

Related Version The full version of the paper (containing all omitted proofs) is [9] available at https://eccc.weizmann.ac.il/report/2019/002/.

Funding *Alexander S. Kulikov*: The results presented in Section 3 are supported by Russian Science Foundation (18-71-10042).
Vladimir Podolskii: The results presented in Section 4 are supported by Russian Science Foundation (16-11-10252).

Acknowledgements We thank Paweł Gawrychowski for pointing us out to the paper [3]. We thank Alexey Talambutsa for fruitful discussions on the theory of semigroups.

1 Introduction

1.1 Problem Statement and New Results

Let $A \in \{0,1\}^{n\times n}$ be a matrix with z zeroes and u ones, and $x = (x_1, \ldots, x_n)$ be an n-dimensional vector of formal variables over a semigroup $(S, \circ)$. In this paper, we study the complexity of the *linear operator* Ax, i.e., how many semigroup operations are required to compute a vector whose i-th element is

$$\sum_{1\le j\le n \bigwedge A_{ij}=1} x_j$$

where the summation is over the semigroup operation $\circ$.[1] More specifically, we are interested in lower and upper bounds involving z and u. Matrices with $u = O(n)$ are usually called *sparse*, whereas matrices with $z = O(n)$ are called *complements of sparse matrices*. Computing all n outputs of Ax directly (i.e. using the above definition) takes $O(u)$ semigroup operations. The main question studied in this paper is: can Ax be computed using $O(z)$ semigroup operations? Note that it is easy to achieve $O(z)$ complexity if $\circ$ has an inverse. Indeed, in this case Ax can be computed via subtraction: $Ax = (U - \overline{A})x = Ux - \overline{A}x$, where U is the all-ones matrix whose linear operator can be computed trivially using $O(n)$ semigroup operations, and $\overline{A}$ is the complement of A and therefore has only $z = O(n)$ ones.

1.1.1 Commutative Case

Our first main result shows that in the commutative case, complements of sparse matrices can be processed as efficiently as sparse matrices. Specifically, we prove that if the semigroup is commutative, Ax can be computed in $O(z)$ semigroup operations; or, more formally, there exists a circuit of size $O(z)$ that uses $x = (x_1, \ldots, x_n)$ as an input and computes Ax by only applying the semigroup operation $\circ$ (we provide the formal definition of the computational model in Section 2.3). Moreover, the constructed circuits are *uniform* in the sense that they can be generated by an efficient algorithm. Hence, our circuits correspond to an elementary algorithm that uses no tricks like examining the values x_j, i.e., the semigroup operation $\circ$ is applied in a (carefully chosen) order that is independent of the specific input x.

▶ **Theorem 1.** *Let $(S, \circ)$ be a commutative semigroup, and $A \in \{0,1\}^{n\times n}$ be a matrix with $z = \Omega(n)$ zeroes. There exists a circuit of size $O(z)$ that uses a vector $x = (x_1, \ldots, x_n)$ of formal variables as an input, uses only the semigroup operation $\circ$ at internal gates, and outputs Ax. Moreover, there exists a randomized algorithm that takes the positions of z zeroes of A as an input and outputs such a circuit in time $O(z)$ with probability at least $1 - \frac{O(\log^5 n)}{n}$. There also exists a deterministic algorithm with running time $O(z + n\log^4 n)$.*

We state the result for square matrices to simplify the presentation. Theorem 1 generalizes easily to show that Ax for a matrix $A \in \{0,1\}^{m\times n}$ with $z = \Omega(n)$ zeroes can be computed using $O(m + z)$ semigroup operations. Also, we assume that $z = \Omega(n)$ to be able to state an upper bound $O(z)$ instead of $O(z + n)$. Note that when $z < n$, the matrix A is forced to contain all-one rows that can be computed trivially.

[1] Note that the result of summation is undefined in case of an all-zero row, because semigroups have no neutral element in general. One can trivially sidestep this technical issue by adding an all-one column $n + 1$ to the matrix A, as well as the neutral element x_{n+1} into the vector. Alternatively, we could switch from semigroups to *monoids*, but we choose not to do that, since we have no use for the neutral element and associated laws in the rest of the paper.

The following corollary generalizes Theorem 1 from vectors to matrices.

▶ **Corollary 2.** *Let $(S, \circ)$ be a commutative semigroup. There exists a deterministic algorithm that takes a matrix $A \in \{0,1\}^{n\times n}$ with $z = O(n)$ zeroes and a matrix $B \in S^{n\times n}$ and computes the product AB in time $O(n^2)$.*

1.1.2 Non-commutative Case

As our second main result, we show that *commutativity is essential*: for a faithful non-commutative semigroup S (the notion of faithful non-commutative semigroup is made formal later in the text), the minimum number of semigroup operations required to compute Ax for a matrix $A \in \{0,1\}^{n\times n}$ with $z = O(n)$ zeroes is $\Theta(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function.

▶ **Theorem 3.** *Let $(S, \circ)$ be a faithful non-commutative semigroup, $x = (x_1, \ldots, x_n)$ be a vector of formal variables, and $A \in \{0,1\}^{n\times n}$ be a matrix with $O(n)$ zeroes. Then Ax is computable using $O(n\alpha(n))$ semigroup operations, where $\alpha(n)$ is the inverse Ackermann function. Moreover, there exists a matrix $A \in \{0,1\}^{n\times n}$ with exactly two zeroes in every row such that the minimum number of semigroup operations required to compute Ax is $\Omega(n\alpha(n))$.*

1.2 Motivation

The complexity of linear operators is interesting for many reasons.

Range queries. In the *range queries* problem, one is given a vector $x = (x_1, \ldots, x_n)$ over a semigroup $(S, \circ)$ and multiple queries of the form (l, r), and is required to output the result $x_l \circ x_{l+1} \circ \cdots \circ x_r$ for each query. It is a classical problem in data structures and algorithms with applications in many fields, such as bioinformatics and string algorithms, computational geometry, image analysis, real-time systems, and others. We review some of the less straightforward applications as well as a rich variety of algorithmic techniques for the problem in the full version of the paper [9].

The linear operator problem is a natural generalization of the range queries problem: each row of the matrix A defines a subset of the elements of x that need to be summed up and this subset is not required to be a contiguous range. The algorithms (Theorem 1 and Corollary 2) and hardness results (Theorem 3) for the linear operator problem presented in this paper are indeed inspired by some of the known results for the range queries problem.

Graph algorithms. Various graph path/reachability problems can be reduced naturally to matrix multiplication. Two classic examples are: (i) the all-pairs shortest path problem (APSP) is reducible to min-plus matrix multiplication, and (ii) the number of triangles in an undirected graph can be found by computing the third power of its adjacency matrix. It is natural to ask what happens if a graph has $O(n)$ edges or $O(n)$ anti-edges (as usual, by n we denote the number of nodes). In many cases, an efficient algorithm for sparse graphs ($O(n)$ edges) is straightforward whereas an algorithm with the same efficiency for complements of sparse graphs ($O(n)$ anti-edges) is not. For example, it is easy to solve APSP and triangle counting on sparse graphs in time $O(n^2)$, but achieving the same time complexity for complements of sparse graphs is more complicated. Theorem 1 and Corollary 2 give a black-box way to solve these two problems on complements of sparse graphs in time $O(n^2)$.

Matrix multiplication over semirings. Fast matrix multiplication methods rely essentially on the ring structure of the underlying set of elements. The first such algorithm was given by Strassen, the current record upper bound is $O(n^{2.373})$ [13, 4]. The removal of the inverse operation often drastically increases the complexity of algorithmic problems

over algebraic structures, and even the complexity of standard computational tasks are not well understood over tropical and Boolean semirings (see, e.g. [12, 6]). For various important semirings, we still do not know an $n^{3-\varepsilon}$ (for a constant $\varepsilon > 0$) upper bound for matrix multiplication, e.g., the strongest known upper bound for min-plus matrix multiplication is $n^3/\exp(\sqrt{\log n})$ [12].

The interest in computations over such algebraic structures has recently grew substantially throughout the Computer Science community with the cases of Boolean and tropical semirings being of the main interest (see, for example, [8, 12, 2]). From this perspective, the computation complexity over sparse and complements of sparse matrices is one of the most basic questions. Theorem 1 and Corollary 2 therefore characterise natural special cases when efficient computations are possible.

Functional programming. Algebraic data structures for graphs developed in the functional programming community [10] can be used for representing and processing densely-connected graphs in linear (in the number of vertices) time and memory. As we discuss in the full version of the paper [9], Theorem 1 yields an algorithm for deriving a linear-size algebraic graph representation for complements of sparse graphs.

Circuit complexity. Computing linear operators over a Boolean semiring $(\{0,1\}, \vee)$ is a well-studied problem in circuit complexity. The corresponding computational model is known as *rectifier networks*. An overview of known lower and upper bounds for such circuits is given by Jukna [7, Section 13.6]. Theorem 1 states that very dense linear operators have linear rectifier network complexity.

1.3 Organization

The remaining part of the paper is organized as follows. In Section 2 we introduce necessary definitions. In Section 3 we present the results on commutative case. In Section 4 we present the results on the non-commutative case. Due to the space constraints many proofs are omitted. They can be found in the full version of the paper [9].

2 Background

2.1 Semigroups and Semirings

A *semigroup* $(S, \circ)$ is an algebraic structure, where the operation $\circ$ is *closed*, i.e., $\circ : S \times S \to S$, and *associative*, i.e., $x \circ (y \circ z) = (x \circ y) \circ z$ for all x, y, and z in S. *Commutative* (or *abelian*) semigroups introduce one extra requirement: $x \circ y = y \circ x$ for all x and y in S.

A commutative semigroup $(S, \circ)$ can often be extended to a *semiring* $(S, \circ, \bullet)$ by introducing another associative (but not necessarily commutative) operation $\bullet$ that *distributes* over $\circ$, that is

$$x \bullet (y \circ z) = (x \bullet y) \circ (x \bullet z)$$

$$(x \circ y) \bullet z = (x \bullet z) \circ (y \bullet z)$$

hold for all x, y, and z in S. Since $\circ$ and $\bullet$ behave similarly to numeric addition and multiplication, it is common to give $\bullet$ a higher precedence to avoid unnecessary parentheses, and even omit $\bullet$ from formulas altogether, replacing it by juxtaposition. This gives a terser and more convenient notation, e.g., the left distributivity law becomes: $x(y \circ z) = xy \circ xz$. We will use this notation, insofar as this does not lead to ambiguity. See the full version of the paper [9] for an overview of commonly used semigroups and semirings.

2.2 Range Queries Problem and Linear Operator Problem

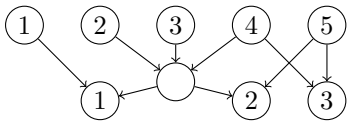
In the *range queries problem*, one is given a sequence $x_1, x_2, \ldots, x_n$ of elements of a fixed semigroup $(S, \circ)$. Then, a *range query* is specified by a pair of indices (l, r), such that $1 \leq l \leq r \leq n$. The answer to such a query is the result of applying the semigroup operation to the corresponding range, i.e., $x_l \circ x_{l+1} \circ \cdots \circ x_r$. The range queries problem is then to simply answer all given range queries. There are two regimes: online and offline. In the *online regime*, one is given a sequence of *values* $x_1 = v_1, x_2 = v_2, \ldots, x_n = v_n$ and is asked to preprocess it so that to answer efficiently any subsequent query. By "efficiently" one usually means in time independent of the length of the range (i.e., $r - l + 1$, the time of a naive algorithm), say, in time $O(\log n)$ or $O(1)$. In this paper, we focus on the *offline* version, where one is given a sequence together with all the queries, and are interested in the minimum number of semigroup operations needed to answer all the queries. Moreover, we study a more general problem: we assume that $x_1, \ldots, x_n$ are formal variables rather than actual semigroup values. That is, we study the *circuit size* of the corresponding computational problem.

The *linear operator* problem generalizes the range queries problem: now, instead of contiguous ranges one wants to compute sums over arbitrary subsets. These subsets are given as rows of a 0/1-matrix A.

2.3 Circuits

We assume that the input consists of n formal variables $\{x_1, \ldots, x_n\}$. We are interested in the minimum number of semigroup operations needed to compute all given words $\{w_1, \ldots, w_m\}$ (e.g., for the range queries problem, each word has a form $x_l \circ x_{l+1} \circ \cdots \circ x_r$). We use the following natural *circuit* model. A circuit computing all these queries is a directed acyclic graph. There are exactly n nodes of zero in-degree. They are labelled with $\{1, \ldots, n\}$ and are called *input gates*. All other nodes have positive in-degree and are called *gates*. Finally, some m gates have out-degree 0 and are labelled with $\{1, \ldots, m\}$; they are called *output gates*. The *size* of a circuit is its number of edges (also called *wires*). Each gate of a circuit computes a word defined in a natural way: input gates compute just $\{x_1, \ldots, x_n\}$; any other gate of in-degree r computes a word $f_1 \circ f_2 \circ \cdots \circ f_r$ where $\{f_1, \ldots, f_r\}$ are words computed at its predecessors (therefore, we assume that there is an underlying order on the incoming wires for each gate). We say that the circuit computes the words $\{w_1, \ldots, w_m\}$ if the words computed at the output gates are equivalent to $\{w_1, \ldots, w_m\}$ over the considered semigroup.

For example, the following circuit computes range queries $(l_1, r_1) = (1, 4)$, $(l_2, r_2) = (2, 5)$, and $(l_3, r_3) = (4, 5)$ over inputs $\{x_1, \ldots, x_5\}$ or, equivalently, the linear operator Ax where the matrix A is given below.



$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

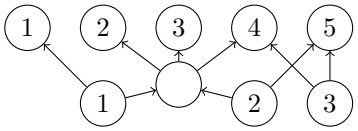
For a 0/1-matrix A, by $C(A)$ we denote the minimum size of a circuit computing the linear operator Ax.

A *binary circuit* is a circuit having no gates of fan-in more than two. It is not difficult to see that any circuit can be converted into a binary circuit of size at most twice the size of the original circuit. For this, one just replaces every gate of fan-in k, for $k > 2$, by a binary tree with $2k - 2$ wires (such a tree contains k leaves hence $k - 1$ inner nodes and $2k - 2$ edges). In the binary circuit the number of gates does not exceed its size (i.e., the number of wires). And the number of gates in a binary circuit is exactly the minimum number of semigroup operations needed to compute the corresponding function.

We call a circuit C computing A *regular* if for every pair (i, j) such that $A_{ij} = 1$, there exists exactly one path from the input j to the output i. A convenient property of regular circuits is the following observation.

▶ **Observation 1.** *Let C be a regular circuit computing a 0/1-matrix A over a commutative semigroup. Then, by reversing all the wires in C one gets a circuit computing A^T.*

Instead of giving a formal proof, we provide an example of a reversed circuit from the example given above. It is because of this observation that we require circuit outputs to be gates of out-degree zero (so that when reversing all the wires the inputs and the outputs exchange places).



$$A^T = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

3 Commutative Case

This section is devoted to the proofs of Theorem 1 and Corollary 2. We start by proving two simpler statements to show how commutativity is important.

▶ **Lemma 4.** *Let S be a semigroup (not necessarily commutative) and let $A \in \{0,1\}^{n\times n}$ contain at most one zero in every row. Then $C(A) = O(n)$.*

Proof. To compute the linear operator Ax, we first precompute all prefixes and suffixes of $x = (x_1, \ldots, x_n)$. Concretely, let $p_i = x_1 \circ x_2 \circ \cdots \circ x_i$. All p_i's can be computed using $(n-1)$ binary gates as follows:

$$p_1 = x_1, p_2 = p_1 \circ x_2, p_3 = p_2 \circ x_3, \ldots, p_i = p_{i-1} \circ x_i, \ldots, p_n = p_{n-1} \circ x_n.$$

Similarly, we compute all suffixes $s_j = x_j \circ x_{j+1} \cdots \circ x_n$ using $(n-1)$ binary gates. From these prefixes and suffixes all outputs can be computed as follows: if a row of A contains no zeroes, the corresponding output is p_n; otherwise if a row contains a zero at position i, the output is $p_{i-1} \circ s_{i+1}$ (for $i = 1$ and $i = n$, we omit the redundant term). ◀

In the rest of the section, we assume that the underlying semigroup is commutative. Allowing at most two zeroes per row already leads to a non-trivial problem. We give only a sketch of the solution below, since we will further prove a more general result. It is interesting to compare the following lemma with Theorem 3 that states that in the non-commutative setting matrices with two zeroes per row are already hard.

▶ **Lemma 5.** *Let $A \in \{0,1\}^{n\times n}$ contain at most two zeroes in every row. Then $C(A) = O(n)$.*

Proof sketch. Consider the following undirected graph: the set of nodes is $\{1, 2, \ldots, n\}$; two nodes i and j are joined by an edge if there is a row having zeroes in columns i and j. In the worst case (all rows are different and contain exactly two zeroes), the graph has exactly n edges and hence it contains a cut (L, R) of size at least $n/2$. This cut splits the columns of the matrix into two parts (L and R). Now let us also split the rows into two parts: the top part T contains all columns that have exactly one zero in each L and R; the bottom part B

contains all the remaining rows. What is nice about the top part of the matrix $(T \times (L \cup R))$ is that it can be computed by $O(n)$ gates (using Lemma 4). For the bottom part, let us cut all-1 columns out of it and make a recursive call (note that this requires the commutativity). The corresponding recurrence relation is $T(n) \leq cn + T(n/2)$ for a fixed constant c, implying $T(n) = O(n)$, and hence $C(A) = O(n)$. ◀

We now state a few auxiliary lemmas that will be used as building blocks in the proof of Theorem 1.

▶ **Lemma 6.** *There exists a binary regular circuit of size $O(n \log n)$ such that any range can be computed in a single additional binary gate using two gates of the circuit. It can be generated in time $O(n \log n)$.*

▶ **Lemma 7.** *There exists a binary regular circuit of size $O(n)$ such that any range of length at least $\log n$ can be computed in two binary additional gates from the gates of the circuit. It can be generated by an algorithm in time $O(n)$.*

▶ **Lemma 8.** *Let $m \leq n$ and $A \in \{0,1\}^{m \times n}$ be a matrix with $z = \Omega(n)$ zeroes and at most $\log n$ zeroes in every row. There exists a circuit of size $O(z)$ computing Ax. Moreover, there exists a randomized $O(z)$ time algorithm that takes as input the positions of z zeros and outputs a circuit computing Ax with probability at least $1 - \frac{O(\log^5 n)}{n}$. There also exists a deterministic algorithm with running time $O(n \log^4 n)$.*

Proof of Theorem 1. Denote the set of rows and the set of columns of A by R and C, respectively. Let $R_0 \subseteq R$ be all the rows having at least $\log n$ zeroes and $R_1 = R \setminus R_0$. Every row of A can be decomposed into (maximal) contiguous ranges of ones. We will call them simply ranges of A. We will compute all of them. From these ranges, it takes $O(z)$ additional binary gates to compute all the outputs.

We compute the matrices $R_0 \times C$ and $R_1 \times C$ separately. The main idea is that $R_0 \times C$ is easy to compute because it has a small number of rows (at most $z/\log n$), while $R_1 \times C$ is easy to compute because it has a small number of zeroes in every row (at most $\log n$).

The matrix $R_1 \times C$ can be computed using Lemma 8. To compute $R_0 \times C$, it suffices to compute $C \times R_0$ by a regular circuit, thanks to the Observation 1. Let $|R_0| = t$. Clearly, $t \leq z/\log n$. Using Lemma 6, one can compute all ranges of $C \times R_0$ by a circuit of size

$$O(t \log t + z) = O\left(\frac{z}{\log n} \cdot \log z + z\right) = O(z + n) = O(z)\,,$$

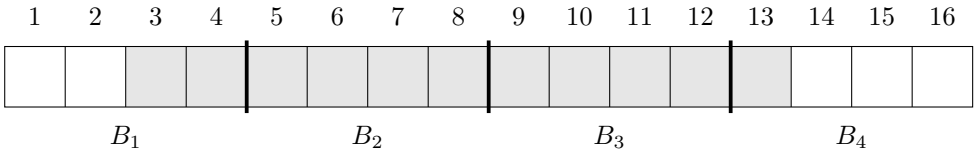
since $z = O(n^2)$.

The algorithm for generating the circuit is just a combination of the algorithms from Lemmas 6 and 8. ◀

Proof of Lemma 6. We adopt the divide-and-conquer construction by Alon and Schieber [1]. Split the input range $(1, n)$ into two half-ranges of length $n/2$: $(1, n/2)$ and $(n/2 + 1, n)$. Compute all suffixes of the left half and all prefixes of the right half. Using these precomputed suffixes and prefixes one can answer any query (l, r) such that $l \leq n/2 \leq r$ in a single additional gate. It remains to be able to answer queries that lie entirely in one of the halves. We do this by constructing recursively circuits for both halves. The resulting recurrence relation $T(n) \leq 2T(n/2) + O(n)$ implies that the resulting circuit has size at most $O(n \log n)$. ◀

Proof of Lemma 7. We use the block decomposition technique for constructing the required circuit. Partition the input range $(1, n)$ into $n/\log n$ ranges of length $\log n$ and call them blocks. Compute the range corresponding to each block (in total size $O(n)$). Build a circuit from Lemma 6 on top of these blocks. The size of this circuit is $O(n)$ since the number of blocks is $n/\log n$. Compute all prefixes and all suffixes of every block. Since the blocks partition the input range $(1, n)$, this also can be done with an $O(n)$ size circuit.

Consider any range of length at least $\log n$. Note that it cannot lie entirely inside the block. Hence, any such range can be decomposed into three subranges: a suffix of a block, a range of blocks, and a prefix of a block (where any of the three components may be empty). For example, for $n = 16$, a range $(3, 13)$ is decomposed into a suffix $(3, 4)$ of the first block, a range $(2, 3)$ of blocks (B_1, B_2, B_3, B_4), and a prefix $(13, 13)$ of the last block:



It remains to note that all these three components are already precomputed. ◀

Proof of Lemma 8. All the z zeroes of A break its rows into ranges. Let us call a range *short* is its length is at most $\log n$. We will show that it is possible to permute the columns of A so that the total length of all short ranges is at most $O(\frac{n}{\log n})$. Then, all such short ranges can be computed by a circuit of size $O(\frac{\log n}{n} \cdot n) = O(n) = O(z)$. All the remaining ranges can be computed by a circuit of size $O(n)$ using Lemma 7.

It is easy to construct the required permutation randomly. For this, one just estimates the expected total length of all short ranges in a random permutation. It is then possible to derandomize this approach using a greedy algorithm. We provide all formal details the full version of the paper [9]. ◀

Proof of Corollary 2. One deterministically generates a circuit for A of size $O(n)$ in time $O(n \log^4 n) = O(n^2)$ by Theorem 1. This circuit can be used to multiply A by any column of B in time $O(n)$. For this, one constructs a topological ordering of the gates of the circuits and computes the values of all gates in this order. Hence, AB can be computed in time $O(n^2)$. ◀

4 Non-commutative Case

In the previous section, we have shown that for commutative semigroups dense linear operators can be computed by linear size circuits. A closer look at the circuit constructions reveals that we use commutativity crucially: it is important that we may reorder the columns of the matrix (we do this in the proof of Lemma 8). In this section, we show that this trick is unavoidable: for non-commutative semigroups, it is not possible to construct linear size circuits for dense linear operators. Namely, we prove Theorem 3.

▶ **Theorem 3.** *Let $(S, \circ)$ be a faithful non-commutative semigroup, $x = (x_1, \ldots, x_n)$ be a vector of formal variables, and $A \in \{0, 1\}^{n \times n}$ be a matrix with $O(n)$ zeroes. Then Ax is computable using $O(n\alpha(n))$ semigroup operations, where $\alpha(n)$ is the inverse Ackermann function. Moreover, there exists a matrix $A \in \{0, 1\}^{n \times n}$ with exactly two zeroes in every row such that the minimum number of semigroup operations required to compute Ax is $\Omega(n\alpha(n))$.*

4.1 Faithful semigroups

We consider computations over general semigroups that are not necessarily commutative. In particular, we will establish lower bounds for a large class of semigroups and our lower bound does not hold for commutative semigroups. This requires a formal definition that captures semigroups with rich enough structure and in particular requires that a semigroup is substantially non-commutative.

Previously lower bounds in the circuit model for a large class of semigroups were known for the Range Queries problem [14, 3]. These result are proven for a large class of commutative semigroups that are called *faithful* (we provide a formal definition below). Since we are dealing with non-commutative case we need to generalize the notion of faithfulness to non-commutative semigroups.

To provide formal definition of faithfulness it is convenient to introduce the following notation. Suppose $(S, \circ)$ is a semigroup. Let $X_{S,n}$ be a semigroup with generators $\{x_1, \ldots, x_n\}$ and with the equivalence relation consisting of identities in variables $\{x_1, \ldots, x_n\}$ over $(S, \circ)$. That is, for two words W and W' in the alphabet $\{x_1, \ldots, x_n\}$ we have $W \sim W'$ in $X_{S,n}$ iff no matter which elements of the semigroup S we substitute for $\{x_1, \ldots, x_n\}$ we obtain a correct equation over S. In particular, note that if S is commutative (respectively, idempotent), then $X_{S,n}$ is also commutative (respectively, idempotent). The semigroup $X_{S,n}$ is studied in algebra under the name of relatively free semigroup of rank n of a variety generated by semigroup S [11]. We will often omit the subscript n and write simply X_S since the number of generators will be clear from the context.

Below we will use the following notation. Let W be a word in the alphabet $\{x_1, \ldots, x_n\}$. Denote by $\texttt{Var}(W)$ the set of letters that are present in W.

We are now ready to introduce the definition of a commutative faithful semigroup.

▶ **Definition 9** ([14, 3]). *A commutative semigroup $(S, \circ)$ is* faithful commutative *if for any equivalence $W \sim W'$ in X_S we have $\texttt{Var}(W) = \texttt{Var}(W')$.*

Note that this definition does not pose any restrictions on the cardinality of each letter in W and W'. This allows to capture in this definition important cases of idempotent semigroups. For example, semigroups $(\{0, 1\}, \vee)$ and $(\mathbb{Z}, \min)$ are commutative faithful.

We need to study the non-commutative case, and moreover, our results establish the difference between commutative and non-commutative cases. Thus, we need to extend the notion of faithfulness to non-commutative semigroups to capture their non-commutativity in the whole power. At the same time we would like to keep the case of idempotency. We introduce the notion of faithfulness for the non-commutative case inspired by the properties of free idempotent semigroups [5]. To introduce this notion we need several definitions.

The *initial mark* of W is the letter that is present in W such that its first appearance is farthest to the right. Let U be the prefix of W consisting of letters preceding the initial mark. That is, U is the maximal prefix of W with a smaller number of generators. We call U the *initial* of W. Analogously we define the *terminal mark* of W and the *terminal* of W.

▶ **Definition 10.** *We say that a semigroup X with generators $\{x_1, \ldots, x_n\}$ is* strongly non-commutative *if for any words W and W' in the alphabet $\{x_1, \ldots, x_n\}$ the equivalence $W \sim W'$ holds in X only if the initial marks of W and W' are the same, terminal marks are the same, the equivalence $U \sim U'$ holds in X, where U and U' are the initials of W and W', respectively, and the equivalence $V \sim V'$ holds in X, where V and V' are the terminals of W and W', respectively.*

In other words, this definition states that the first and the last occurrences of generators in the equivalence separates the parts of the equivalence that cannot be affected by the rest of the generators and must therefore be equivalent themselves. We also note that this definition exactly captures the idempotent case: for a free idempotent semigroup the condition in this definition is "if and only if"[5].

▶ **Definition 11.** *A semigroup* $(S, \circ)$ *is* faithful non-commutative *if* X_S *is strongly non-commutative.*

We note that this notion of faithfulness is relatively general and is true for semigroups $(S, \circ)$ with considerable degree of non-commutativity in their structure. It clearly captures free semigroups with at least two generators. It is also easy to see that the requirements in Definition 11 are satisfied for the free idempotent semigroup with n generators (if S is idempotent, then $X_{S,n}$ is also clearly idempotent and no other relations are holding in $X_{S,n}$ since we can substitute generators of S for $x_1, \ldots, x_n$).

Next we observe some properties of strongly non-commutative semigroups that we need in our constructions.

▶ **Lemma 12.** *Suppose X is strongly non-commutative. Suppose the equivalence $W \sim W'$ holds in X and $|\mathtt{Var}(W)| = |\mathtt{Var}(W')| = k$. Suppose U and U' are minimal (maximal) prefixes of W and W' such that $|\mathtt{Var}(U)| = |\mathtt{Var}(U')| = l \leq k$. Then the equivalence $U \sim U'$ holds in X. The same is true for suffixes.*

Proof. The proof is by induction on the decreasing l. Consider the maximal prefixes first. For $l = k$ and maximal prefixes we just have $U = W$ and $U' = W'$. Suppose the statement is true for some l, and denote the corresponding prefixes by U and U', respectively. Then note that the maximal prefixes with $l - 1$ variables are initials of U and U'. And the statement follows by Definition 10.

The proof of the statement for minimal prefixes is completely analogous. Note that on the step of induction the prefixes differ from the previous case by one letter that are initial marks of the corresponding prefixes. So these additional letters are also equal by the Definition 10.

The case of suffixes is completely analogous. ◀

The next lemma is a simple corollary of Lemma 12.

▶ **Lemma 13.** *Suppose X is strongly non-commutative. Suppose $W \sim W'$ holds in X. Let us write down the letters of W in the order in which they appear first time in W when we read it from left to right. Let's do the same for W'. Then we obtain exactly the same sequences of letters. The same is true if we read the words from right to left.*

4.2 Proof Strategy

We now proceed to the proof of Theorem 3. The upper bound follows easily by a naive algorithm: split all rows of A into ranges, compute all ranges by a circuit of size $O(n\alpha(n))$ using Yao's construction [14], then combine ranges into rows of A using $O(n)$ gates.

Thus, we focus on lower bounds. We will view the computation of the circuit as a computation in a strongly non-commutative semigroup $X = X_S$.

We will use the following proof strategy. First we observe that it is enough to prove the lower bound for the case of idempotent strongly non-commutative semigroups X. Indeed, if X is not idempotent, we can factorize it by idempotency relations and obtain a strongly non-commutative idempotent semigroup X_{id}. A lower bound for the case of X_{id} implies lower bound for the case of X. We provide a detailed explanation in the full version of the paper [9].

Hence, from this point we can assume that X is idempotent and strongly non-commutative. Next for idempotent case we show that our problem is equivalent to the commutative version of the range query problem.

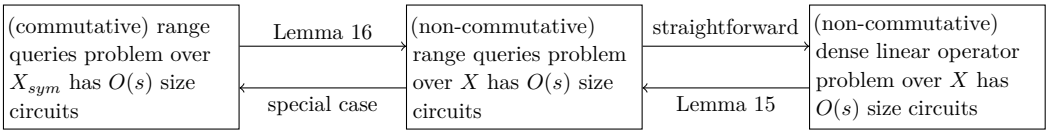
For a semigroup X with generators $\{x_1, \ldots, x_n\}$ denote by X_{sym} its factorization under commutativity relations $x_i x_j \sim x_j x_i$ for all i, j. Note that if X is idempotent and strongly non-commutative, then X_{sym} is just the semigroup in which $W \sim W'$ iff $\texttt{Var}(W) = \texttt{Var}(W')$ (this is free idempotent commutative semigroup).

▶ **Theorem 14.** *For an idempotent strongly non-commutative X and for any $s = \Omega(n)$ we have that (commutative) range queries problem over X_{sym} has size $O(s)$ circuits iff (non-commutative) dense linear operator problem over X has size $O(s)$ circuits.*

Using this theorem, it is straightforward to finish the proof of Theorem 3. Indeed, by Theorem 14 if non-commutative dense linear operator problem has size s circuit, then the commutative range queries problem also does. However, for the latter problem it is proved by Chazelle and Rosenberg [3] that $s = \Omega(n\alpha(n))$. Moreover, in our construction for the proof of Theorem 14 it is enough to consider dense linear operators with exactly two zeroes in every row. From this the second part of Theroem 3 follows.

Note that for the proof of Theorem 3 only one direction of Theorem 14 is needed. However, we think that the equivalence in Theorem 14 might be of independent interest, so we provide the proof for both directions.

Thus, it remains to prove Theorem 14. We do this by showing the following equivalences for any $s = \Omega(n)$.



Note that two of the reductions on this diagram are trivial. The other two are formulated in the following lemmas.

▶ **Lemma 15.** *If the (non-commutative) dense linear operator problem over X has size s circuit then the (non-commutative) range queries problem over X has size $O(s)$ circuit.*

▶ **Lemma 16.** *If the (commutative) version of the range queries problem over X_{sym} has size s circuits then the (non-commutative) version over X also does.*

5 Open Problems

There are two natural problems left open.

1. Design a deterministic $O(z)$ time algorithm for generating a circuit in the commutative case. For this, it suffices to design an $O(n)$ deterministic algorithm for the following problem: given a list of positions of n zeroes of an $n \times n$ 0/1-matrix with at most $\log n$ zeroes in every row, permute its columns so that the total length of all segments of length at most $O(\log n)$ is $O(\frac{n}{\log n})$.
2. Determine the asymptotic complexity of the linear operator in terms of the number of zeroes in the non-commutative case.

References

1 Noga Alon and Baruch Schieber. Optimal preprocessing for answering on-line product queries. Technical report, Tel Aviv University, 1987.

2 Peter Butkovič. *Max-linear Systems: Theory and Algorithms*. Springer, 2010.

3 Bernard Chazelle and Burton Rosenberg. The complexity of computing partial sums off-line. *Int. J. Comput. Geometry Appl.*, 1(1):33–45, 1991. doi:10.1142/S0218195991000049.

4 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

5 J. A. Green and D. Rees. On semi-groups in which $x^r = x$. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(1):35–40, 1952. doi:10.1017/S0305004100027341.

6 Dima Grigoriev and Vladimir V. Podolskii. Complexity of Tropical and Min-plus Linear Prevarieties. *Computational Complexity*, 24(1):31–64, 2015. doi:10.1007/s00037-013-0077-5.

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

8 Stasys Jukna. Tropical Complexity, Sidon Sets, and Dynamic Programming. *SIAM J. Discrete Math.*, 30(4):2064–2085, 2016. doi:10.1137/16M1064738.

9 Alexander S. Kulikov, Ivan Mikhailin, Andrey Mokhov, and Vladimir V. Podolskii. Complexity of Linear Operators. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:2, 2019. URL: https://eccc.weizmann.ac.il/report/2019/002.

10 Andrey Mokhov. Algebraic graphs with class (functional pearl). In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell*, pages 2–13. ACM, 2017.

11 H. Neumann. *Varieties of Groups*. Ergebnisse der Mathematik und ihrer Grenzgebiete. 2. Folge. Springer Berlin Heidelberg, 2012. URL: https://books.google.ru/books?id=VaMjCQAAQBAJ.

12 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 664–673, 2014. doi:10.1145/2591796.2591811.

13 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith–Winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.

14 Andrew Chi-Chih Yao. Space-Time Tradeoff for Answering Range Queries (Extended Abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 128–136. ACM, 1982. doi:10.1145/800070.802185.