Range Queries and Dense Linear Operators

21 июня 2018 г.

Аннотация

We revisit the classical problem of computing range queries over an idempotent semigroup and its natural special case, computing a dense linear operator. Specifically, we are interested in the minimum number of semigroup operations needed to answer all the queries. For commutative semigroups, we show that the special case is strictly easier: while for range queries a superlinear lower bound is known, a dense linear operator can be computed by linear size circuits. We then study the role of commutativity in these problems: it turns out to be unimportant for range queries, but crucial for dense operators. Moreover, without commutativity both problems are equivalent.

Андрей, допиши про applications, пожалуйста

Содержание

1 General Setting

1.1 Range Queries Problem

Range queries is a classical problem in data structures and algorithms that is stated as follows. For a fixed semigroup with operator $\circ$, one is given a sequence $x_1, x_2, \ldots, x_n$ of group elements. Then,

a range query is specified by two indices (l, r) such that $1 \le l \le r \le n$. The answer to such a query is the result of applying the semigroup operator 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 $x_1, x_2, \ldots, x_n$ and is asked to preprocess it into a data structure so that to answer efficiently any subsequent query. By saying efficiently one usually means in time independent of the length of the range (i.e., $j-i+1$, the time of naive answer), say, in time $O(\log n)$ or $O(1)$. In this paper, we focus on the offline version of range queries and are interested in the minimum number of semigroup operations needed to answer all the queries.

tell about dense linear operators

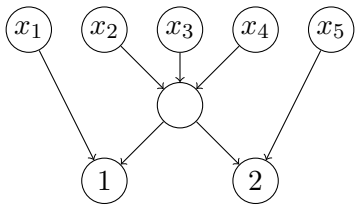
1.2 Applications

Andrey, convince the reader that the problem is important. Show many applications. Show various semigroups arising in real life: $(\mathbb{Z}, +)$, $(\mathbb{Z}, \min)$, Boolean, etc. Emphasize applications of dense linear operators. One relevant link: `http://www.leptonica.com/grayscale-morphology.html`

1.3 Computational Model

We assume that the input consists of n formal variables $x_1, x_2, \ldots, x_n$ (rather than actual group elements). We are interested in the minimum number of semigroup operations needed to answer all the queries. Hence, we use the following natural *circuit* model. For a given set of queries $(l_1, r_1), \ldots, (l_m, r_m)$, a circuit computing all these queries is a directed acyclic graph. There are exactly n nodes of zero in-degree. They are labeled with $x_1, \ldots, x_n$ 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 labeled as *output gates*. The *size* of a circuit is its number of edges (also called *wires*). Each gate of a circuit computes a function defined in a natural way: input gates compute just $x_1, \ldots, x_n$; any other gate of in-degree r computes a function $f_1 \circ f_2 \circ \cdots \circ f_r$ where $f_1, \ldots, f_r$ are functions computed at its predecessors (therefore, we assume that there is an underlying order on the incoming wires for each gate).

For example, the following circuit computes range queries $(l_1, r_1) = (1, 4)$ and $(l_2, r_2) = (2, 5)$ over $x_1, \ldots, x_5$.



mention rectifier networks

1.4 Overview of Known Approaches

Sasha: prefix sums, sparse table, segment trees, Yao, BFC, etc

1.5 Overview of New Results

Sasha, write this

2 Commutative Semigroups

Ваня, давай!

2.1 Matrices with Constant Maximum Number of Zeros in Each Row

2.2 Matrices with Constant Average Number of Zeros in Each Row

3 Non-commutative Semigroups

Володя, жарь!

Theorem 1. *Intervals are computable by $O(n)$-size circuit iff non-commutative dense matrices are computable by $O(n)$-size circuit.*

To show the theorem we introduce an intermediate problem: computing non-commutative intervals by $O(n)$-size circuit.

Clearly, this problem subsumes both of our problems. Indeed, if we can compute non-commutative intervals, we can compute commutative intervals by the same circuit. On the other hand, if we can compute non-commutative intervals, then given non-commutative dense matrix we can split it into intervals, compute them separately, then join them together in $O(n)$.

Thus, it remains to show the following two lemmas.

Lemma 1. *If we can compute non-commutative dense matrices by a linear size circuit, we can also compute non-commutative intervals.*

Lemma 2. *If we can compute commutative intervals by a linear size circuit, we can also compute non-commutative intervals.*

4 Proof of Lemma 1

The intuition why this might be true is that it seems that the best way to compute rows of dense matrix is to construct clauses with the correct order of variables. If this is true, then the idea is that we can extend each interval by variables on both sides (with small gaps on each side of the interval), thus obtaining a dense matrix. Then we can try to extract the computation of the intervals from the computation of the dense matrix. Since all rows of the matrix are hopefully computed by consecutively adding the variables in the correct order, we might be able to do that.

But some analysis of all these things is needed since idempotency is tricky. For example, it is possible to simulate commutativity: if we have the product xy and would like to have yx we can just multiply by x from the left and by y from the right. Then we have $xyxy = xy$. We can use this

to put some variables inside of already produced products. If we have xz and would like to obtain xyz we can just multiply by xyz from the left and by xyz from the right. Then we have

$$(xyz)xz(xyz) = xy(zxzx)yz = xy(zx)yz = xyz.$$

This is not extremely impressive, since to obtain xyz we multiply by xyz, but the point is that this is possible in principle. So, to prove the lemma some more work is needed.

We need the following notations and facts on free idempotent semigroups [1].

Let W be a word in a free idempotent semigroup with generators $\{a_1, \ldots, a_n\}$ (we will also call them variables and letters). Denote by $\texttt{Var}(W)$ the set of generators that are present in W. The initial mark of W is the generator 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 smaller number of generators. We call U the initial of W. Analogously we define terminal mark of W and the terminal of W.

Lemma 3 ([1])**.** *If $W \sim W'$ in the semigroup, then their initial mark are the same, terminal marks are the same, $U \sim U'$ in the semigroup where U and U' are initials of W and W' respectively, $V \sim V'$ in the semigroup where V and V' are terminals of W and W' respectively.*

Lemma 4. *Suppose $W \sim W'$ in the semigroup and W and W' consists of k generators. Suppose U and U' are minimal (maximal) prefixes consisting of $l \leq k$ generators. Then $U \sim U'$.*

Доказательство. 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, denote the corresponding prefixes by U and U' respectively. Then note that the maximal prefixes with $l-1$ letters are initials of U and U'. And the statement follows by Lemma 3.

The proof of the statement for minimal prefixes is completely analogous. Note that on the step of induction the prefixes differs from the previous case by one letter that are initial marks of the corresponding prefixes. So these additional letters are also equal by Lemma 3. □

The next lemma is a simple corollary of Lemma 4.

Lemma 5. *Suppose $W \sim W'$. 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.

So, we consider the free idempotent semigroup with generators $\{a_1, \ldots, a_n\}$. We consider the generators to be ordered from a_1 to a_n in the increasing order. We want to compute $B \cdot \vec{a}$, where $\vec{a} = (a_1, \ldots, a_n)$ and B is a boolean matrix.

We will show that any circuit computing $B \cdot \vec{a}$ can be reconstructed into another circuit that we will call *an interval circuit* without increase in the size of the circuit. In this circuit we require that each gate computes a word that is equivalent to a word consisting of increasing sequence of letters. Note that as a consequence we have that if in an interval circuit we multiply two gates f and h, then the increasing sequences of letters computed by f and h are matching, that is some suffix of f is equal to some suffix of h. Otherwise, the product is not equal to an increasing sequence of variables.

Once we show that any circuit solving our problem can be reconstructed into an interval one, it is easy to reduce an interval problem to our problem. Note, that given a circuit computing $B \cdot \vec{a}$ for

a super-dense matrix B we can construct a circuit computing all intervals of this matrix. Indeed, first reconstruct a circuit into an interval one. Then, once the interval circuit is trying to multiply intervals that have gaps between them, we just not multiply them. Later on if we try to do something with the product we have not computed we use the left of its inputs in case we try to multiply from the left, and the right input if we are multiplying from the right. So, if we need to solve an interval problem, for each interval we skip one variable on each side and add all other variables to the interval. This turn our problem into the super-dense matrix. We compute this matrix, then deduce the computation of intervals as described above.

To reconstruct a circuit into a linear one we need to introduce some notation. Consider a circuit C and its gate g. We will identify gates and words that they are computing. We will treat the results of the computation of the gates as words to which gates apply concatenation operation. That is, we consider these words before we apply any equivalences in the semigroup to them. Suppose letter a_i is contained in g. We say that a_i is *good* in g if there is a path in C from g to some output, on which the word is never multiplied from the left by words with letters greater or equal to a_i.

Note that if a_i and $a_{i'}$ are contained in g, $i < i'$ and a_i is good in g, then $a_{i'}$ is also good in g. That is, the set of all good letters in g is closed upwards.

Consider the largest good letter in g (if there is one), denote it by a_k (if there are good letters in g, then a_k is actually just the largest letter in g). Consider the first occurrence of a_k in g.

Claim 1. *All first occurrences of other good letters in g must be to the left of a_k.*

Доказательство. Suppose that some good letter a_i has the first occurrence to the right of a_k. Consider an output c such that there is a path from g to c and along this path there are no multiplications of g from the left by words containing letters greater than a_i. Then we have $c \sim LgR$, where all letters of L are smaller then a_i. Then in c letter a_i appears before a_k when we read from left to right, and in RgL we have that a_k appears before a_i. This contradicts Lemma 5. □

Consider for the gate g two words, $\mathtt{MIN}_g$ and $\mathtt{MAX}_g$. Both this words are products of variables in the increasing order: $\mathtt{MIN}_g$ is the product of good letters of g in the increasing order, $\mathtt{MAX}_g$ is the product (in the increasing order) of all letters that has first occurrences before a_k. Note that $\mathtt{MIN}_g$ is a suffix of $\mathtt{MAX}_g$. If there are no good letters in g we just let $\mathtt{MIN}_g = \mathtt{MAX}_g = \lambda$ (the empty word).

For the word W that has the form of the product of variables in the increasing order we call a_j a *gap variable* if it is not contained in W and there are a_i and a_l contained in W such that $i < j < l$.

We will show how given a circuit C to construct an interval circuit C' that for each gate g of C computes some intermediate product P_g between $\mathtt{MIN}_g$ and $\mathtt{MAX}_g$: we will have that $\mathtt{MIN}_g$ is a suffix of P_g and P_g is a suffix of $\mathtt{MAX}_g$. The size of C' is at most the size of C. Note that for an output gate g we have $\mathtt{MIN}_g = \mathtt{MAX}_g = g$, so the circuit C' computes the correct outputs.

The construction of C' is by induction on the number of gate. For the case of input gate g everything is obvious: $\mathtt{MAX}_g$ is either λ, or a_j for some variable a_j. Both of these are easy to compute.

For the step of induction consider a gate $g = f \cdot h$ computed as a product of two previous gates.

Consider the good letters in g. If there are none, we can just let $P_g = \lambda$ and there is no computation needed.

If all first occurrences of good letters of g are lying in one of the input gates f and h, then they are good in the corresponding input gate. So we can set P_g to P_f or P_h and there is no computation needed.

The only remaining case is that some good letters have their first occurrence in f and some in h. Then the largest letter a_k of g has the first occurrence in h and all letters of f are smaller than a_k.

Claim 2. *There are no gap letters for $\texttt{MAX}_h$ in f.*

Доказательство. Suppose that some letter a_i in f is a gap letter for $\texttt{MAX}_h$. Consider an output c such that there is a path from g to c and along this path there are no multiplications of g from the left by words containing letters greater than a_k. Then we have $c \sim LgR$, where all letters of L are smaller then a_k. Consider the prefix P of c preceding the letter a_k and the prefix Q of Lg preceding the letter a_k. Then by Lemma 4 we have $P \sim Q$. But then the letters of P and Q appear in the same order if we read the words from right to left. But this is not true (the letters in P are in the decreasing order and in Q the letter a_i is not on its place), so we get a contradiction. □

Claim 3. *There are no gap letters for $\texttt{MAX}_f$ in h.*

Доказательство. Suppose that some letter a_i in h is a gap letter for $\texttt{MAX}_f$. Consider an output c such that there is a path from g to c and along this path there are no multiplications of g from the left by words containing letters greater than a_l, the largest letter of f. Then we have $c \sim LgR$, where all letters of L are smaller then a_l. Consider the prefix P of c preceding the letter a_l and the prefix Q of Lg preceding the letter a_l. Then by Lemma 4 we have $P \sim Q$. But then the letters of P and Q appear in the same order if we read the words from right to left. But this is not true(the letters in P are in the decreasing order and in Q the letter a_i is not on its place), so we get a contradiction. □

Consider P_f and P_h. From two Claims 2 and 3 we know that they are intervals in the same sequence of variables $\texttt{Var}(P_f) \cup \texttt{Var}(P_h)$. We know that the largest letter of P_h is greater than all letters of P_f. Then either P_f is contained in P_h, and then we can let $P_g = P_h$ (it contains all good letters of g), or we have $P_f = PQ$ and $P_h = QR$ for some words P, Q, R. In the latter case we can let $P_g = P_f \cdot P_h = PQQR = PQR$. Clearly, $\texttt{MIN}_g$ is the suffix of P_g and P_g itself is the suffix of $\texttt{MAX}_g$. So, we are done.

5 Proof of Lemma 2

For the proof of this lemma we will show that any computation of commutative intervals can be reconstructed without increase in the number of gates in such a way that each gate computes an interval (still commutatively; let's call this an interval circuit). It is easy to see that then this circuit can be reconstructed as a non-commutative circuit each gate of which computes the same interval with the variables in the right order. Indeed, we need to make sure that each gate computes an interval in such a way that all variables are in the right order and this is easy to do by induction. Each gate computes an OR of two intervals a and b. If one of them is contained in the other, we simplify the circuit, since the gate just computes the same interval as one of its inputs. It is impossible that a and b are non-intersecting and have a gap between them, since then our gate does not compute an interval (in the interval circuit). So, if a and b are non-intersecting, then they are consecutive and we just need to multiply then in the right order. If the intervals are intersecting, we just multiply then in the right order and apply idempotency (like this: $(x_1x_2x_3)(x_2x_3x_4) = x_1(x_2x_3)(x_2x_3)x_4 = x_1x_2x_3x_4$).

Thus it remains to show that each non-commutative circuit can be reconstructed into an interval circuit. For this we will need some notation.

Suppose we have some circuit C. For each gate g denote by $\texttt{left}(g)$ the smallest index of the variable in g (the leftmost variable). Analogously denote by $\texttt{right}(g)$ the largest index of the variable in g. Denote by $\texttt{gap}(g)$ the smallest i such that x_i is not in g, but there are some j, k such that $j < i < k$ and x_j and x_k (the smallest index of the variable that is in the gap in g). Next, fix some ordering of gates in C (the ordering should be proper, that is inputs to any gate should have smaller numbers). Denote by $\texttt{num}(g)$ the number of a gate in this ordering. Finally, by $\texttt{out}(g)$ denote the out-degree of g.

For each gate that computes a non-interval consider the tuple

$$\texttt{tup}(g) = (\texttt{left}(g), \texttt{gap}(g), \texttt{num}(g), -\texttt{out}(g)).$$

For the circuit C consider $\texttt{tup}(C) = \min_g \texttt{tup}(g)$, where the minimum is considered in the lexicographic order and is taken over all non-interval gates. If there are no non-interval gates we let $\texttt{tup}(C) = \infty$. This is our semi-invariant, we will show that if we have a circuits that is not an interval circuit, we can reconstruct it to increase its $\texttt{tup}$ (in the lexicographic order) without increasing its size. Since $\texttt{tup}$ ranges over a finite set, we can reconstruct the circuit repeatedly and end up with an interval circuit.

Now we are ready to describe a reconstruction. Consider a circuit C that is not an interval circuit. And consider a gate g such that $\texttt{tup}(g) = \texttt{tup}(C)$ (it is clearly unique). Denote by a and b two inputs of g. Let $i = \texttt{left}(g)$ and $j = \texttt{gap}(g)$, that is x_i is the variable with the smallest index in g and x_j is the first gap variable of g (it is not contained in g).

The variable x_i is contained in at least one of a and b. Consider the gate among a and b that contains x_i. It also contain all variables between x_i and x_j (but not x_j), since the converse would contradict minimality of g (by the second coordinate of $\texttt{tup}$). But this gate cannot have x_j as a gap variable: it would also contradict minimality of g (by the third coordinate of $\texttt{gap}$). Thus this gate is exactly the interval $[x_i, x_j)$ (by this we denote the product of variables from x_i to x_j excluding x_j). In particular, only one of a and b contains x_i: otherwise they are both $[x_i, x_j)$ and x_j is not a gap variable for g.

From now on we assume that a contains x_i, that is $a = [x_i, x_j)$.

Now we consider all gates $h_1, \ldots, h_k$ that have edges leading from g. Denote by $f_1, \ldots, f_k$ their other inputs. If k is equal to 0, we can remove g and reduce the circuit. Now we consider cases.

Case 1. Suppose that there is l such that $\texttt{left}(f_l) \leq \texttt{left}(g)$. Then f_l contains all variables in $[x_i, x_j)$ (the contrapositive contradicts the minimality of g by the second coordinate of $\texttt{gap}$). Thus f_l contains a. Then, we can restructure the circuit by feeding b to h_l instead of g. This does not change the clause computed by h_l and reduces $\texttt{out}(g)$. Thus $\texttt{tup}(C)$ increases and we are done.

Case 2. Suppose that for all l we have $\texttt{left}(f_l) > \texttt{left}(g)$. Consider l such that f_l has the minimal $\texttt{right}(f_l)$ (if there are several such l pick among them the one with the minimal $\texttt{num}(f_l)$). Now we restructure the circuit in the following way. We feed f_l to g instead of a. We feed a to h_l instead of f_l. We feed h_l to all other h_p's instead of g. It is not hard to see that all these reconstructions are valid, that is, do not create cycles. Note that they might require reordering of the circuit gates, since we create edges between previously incomparable h-gates. But the reording changes only for the gates with $\texttt{num}$ greater than $\texttt{num}(g)$.

Observe, that the circuit still computes the outputs correctly. The changes are in the gates $h_1 \ldots, h_k$ (and also in g, but $h_1, \ldots, h_k$ are all of its outputs). h_l does not change. Other h_p's might

have changed, they now additionally include variables of f_l. But note that all of these variables are in between of $\texttt{left}(h_p)$ and $\texttt{right}(h_p)$, so they must be presented in the output gates connected to h_p anyway.

Now, observe that $\texttt{num}(g)$ has increased (by the first coordinate). There are no new gates with smaller `left`. Among gates with the minimal `left` there are no new gates with smaller `gap`. Among gates with minimal $(\texttt{left}, \texttt{gap})$ all gates have larger `num` then g. Thus $\texttt{tup}(C)$ increased and we are done.

6 Applications

Андрей, фигарь!

Список литературы

[1] 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.