Cell-Probe Lower Bounds from Online Communication Complexity

Josh Alman* Joshua R. Wang† Huacheng Yu‡

April 20, 2017

Abstract

In this work, we introduce an online model for communication complexity. Analogous to how online algorithms receive their input piece-by-piece, our model presents one of the players Bob his input piece-by-piece, and has the players Alice and Bob cooperate to compute a result it presents Bob with the next piece. This model has a closer and more natural correspondence to dynamic data structures than the classic communication models do and hence presents a new perspective on data structures.

We first present a lower bound for the *online set intersection* problem in the online communication model, demonstrating a general approach for proving online communication lower bounds. The online communication model prevents a batching trick that classic communication complexity allows, and yields a stronger lower bound. Then we apply the online communication model to data structure lower bounds by studying the Group Range Problem, a dynamic data structure problem. This problem admits an $O(\log n)$-time worst-case data structure. Using online communication complexity, we prove a tight cell-probe lower bound: spending $o(\log n)$ (even amortized) time per operation results in at best an $\exp(-\delta^2 n)$ probability of correctly answering a $(1/2+\delta)$-fraction of the n queries.

*MIT CSAIL and EECS, jalman@mit.edu. Supported by an NSF Graduate Research Fellowship.

†Stanford University, joshua.wang@cs.stanford.edu. Supported by NSF CCF-1524062 and a Stanford Graduate Fellowship.

‡Stanford University, yuhch123@gmail.com. Supported by NSF CCF-1212372.

1 Introduction

One of the successes of complexity theory has been Yao's cell-probe model [Yao81], a powerful model of computation which captures the difficult aspects of many data structure problems. In this model, the data structure is charged only for the number of cells of memory that it probes, and not for any computation that it does on the contents of those cells. The strength of this model means that lower bounds here apply to most common models of data structure computation as well. The model bears similarities to communication complexity, where the players are charged only for the bits that they send to each other and not for computation they do. It is hence unsurprising that many cell-probe lower bound results are proved by invoking communication complexity [Ajt88, MNSW95, BF02, Păt07, PT11, Yu16, WY16].

Unfortunately, the sheer power offered to the data structure by this model often makes it difficult to prove strong lower bounds. In many cases, a matching lower bound in the cell-probe model may be impossible; it is possible that counting cell probes in lieu of computation time does indeed make several problems easier [LW17]. Partly as a result of this difficulty, only a few techniques are known for proving cell-probe lower bounds. In this paper, we propose a new technique to add to our growing toolbox, and demonstrate its usefulness by using it to prove a lower bound for the following variant of the Partial Sums problem (from e.g. [PD04]):

The problem we consider is the *Group Range Problem* on a group G along with a *binary encoding* of the group elements $e : G \to \{0,1\}^s$. We will be considering this problem in the cell-probe model with cell-size $w = \Theta(\log n)$, and when the group is not too large: $\log |G| = O(w)$. Our data structure should store a sequence of n group elements $a_0, \ldots, a_{n-1}$ while supporting the following operations:

- Update(i, a) sets entry $a_i \leftarrow a$.
- Query(ℓ, r, i) returns the i^{th} bit of the binary encoding of the group product $a_\ell a_{\ell+1} \cdots a_{r-1} a_r$.

Our main result about this problem is a strong lower bound:

Theorem 1.1. *There exists a distribution over n updates and queries for the Group Range Problem with binary encoding of the group elements $e : G \to \{0,1\}^s$, such that for any randomized cell-probe data structure D with word size $w = \Theta(\log n)$, which with probability p answers at least a $(\frac{1}{2} + \delta)$ fraction of queries correctly and spends $\epsilon n \log n$ total running time, we must have $p \leq \exp(-\delta^2 n)$, as long as $s \leq (1+\epsilon)\log|G|$, $\delta^2 \gg \epsilon \geq \Omega(1/\log n)$, and n is sufficiently large.*[1]

Theorem 1.1 settles the trade-off between the running time and the accuracy of the output for the Group Range Problem. There are two regimes: Either we are willing to pay $\Theta(\log n)$ time per operation, in which case, there exists a deterministic worst-case data structure,[2] or else we require the data structure to spend $o(\log n)$ time per operation, in which case, Theorem 1.1 shows that we cannot hope to do much better than outputting a random bit for each query (except for a constant factor improvement in δ). To the best of our knowledge, this is the first tight data structure lower bound in such a high error regime, where a data structure may answer barely more than half the queries correctly, and do so with such small success probability.

Our lower bound is proved by means of a novel model of communication we introduce, called the *online communication model*. In this model, Alice gets an input x, and Bob gets inputs $y_1, \ldots, y_k$ for a parameter k, and there is a function f such that Alice and Bob want to compute $f(x, y_i)$ on all pairs (x, y_i). In usual models of communication, Bob is given all of $y_1, \ldots, y_k$ as input at the beginning of the communication protocol. Instead, in the online model, Bob starts with only y_1, and Alice and Bob need to compute $f(x, y_i)$ before Bob is given y_{i+1}.

[1] In this paper, we use $\exp(f(n))$ to mean $2^{\Theta(f(n))}$.

[2] Any classic data structure for the Partial Sums problem solves our problem.

Our main result about the online communication model is that the online version of the set intersection problem requires more communication than the offline version. Informally, in the online set intersection problem, Alice is given a set $X \subseteq [n]$ of size k, and Bob is given the elements y_i of another set $Y \subseteq [n]$ one by one. The players have to decide whether y_i is in X before Bob receives the next element.

Theorem 1.2 (informal)**.** *When $n \geq k^2$, the online set intersection problem requires $\Omega(k \log \log k)$ bits of total communication.*

Furthermore, our proof shows that deciding whether X and Y are disjoint (whether we have $y_i \notin X$ for all i) requires $\Omega(k \log \log k)$ bits of communication in the online model. In contrast, an $O(k)$-bit protocol exists for the set disjointness problem in the classic communication model, by using a "batching" trick to test all the y_i simultaneously [HW07]. Our lower bound in the online model is tight for large enough k, since there is a simple protocol that solves the problem in $O(k \log \log n)$ bits of communication. See Section 2.2 for more details.

1.1 Our Technique and Related Work

In this subsection, we introduce the new ideas used in the proof of Theorem 1.1. The high-level idea is similar to the previous communication-complexity-based techniques for proving dynamic cell-probe lower bounds [PT11, Yu16, WY16]: first "decompose" the computation into communication games, and show that an efficient data structure induces efficient protocols for these games; then prove communication lower bounds for them. To decompose the computation into communication games, we consider a random sequence of operations consisting of updates and queries, and two consecutive intervals in it. Roughly speaking, in each communication game, the first interval of operations is only revealed to the first player Alice, while the second interval is only revealed to the second player Bob (all other operations are revealed to both players). The goal of the game is to cooperatively answer all queries in the second interval.

The choice of the communication model in this type of proof is crucial. The first step, solving the communication game given a fast data structure, can be done more efficiently if we use a stronger model. The second step, proving the communication lower bounds, is tougher if the model we study is too powerful. Hence, designing the right communication model as the proxy is one of the key ingredients in the proof. In the proof of Theorem 1.1, we analyze our communication game in the *online communication model* (see Section 2.1 for the formal definition). Compared to other models used in previous work to prove cell-probe lower bounds, the online communication model has a more natural correspondence to data structures. A data structure must output the answer to a query before it sees the next operation; similarly, an online communication protocol must compute the current function value before the players receive the next (partial) input. Studying the game in the online communication model allows us to examine the data structure from a more fine-grained view. See Section 3.1.3 for more details on the connection between online communication complexity and dynamic data structures.

More importantly, the communication game induced by our Group Range Problem (for some group G) has a protocol in the offline setting, where both players receive their inputs at once, which is too efficient,[3] precluding a strong enough communication lower bound which is necessary to prove a tight data structure lower bound. Therefore, it is provably impossible to use any of the previous communication models as the proxy — the communication game is simply not "hard" in them. The detailed proof of our main theorem can be found in Section 3.1.

Pătraşcu and Demaine [PD04] proved a (tight) $\Omega(\log n)$ lower bound for the Partial Sums problem, a problem similar to our *Group Range Problem*, where the Query operation returns the entire group product

[3]For some G, Bob has a succinct encoding of his queries. Thus, sending the (compressed) input to Alice allows them to solve the problem more efficiently than the trivial protocol.

rather than a single bit, and no error is allowed in answering queries. Their *information-transfer technique* does not apply directly to our problem, since it relies on the fact that each query outputs many bits and hence reveals a lot of information, and that the data structure has no errors. Their technique was later generalized [PD06] to prove lower bounds for problems with single-bit output, but their argument mostly focuses on the query which the data structure spends the "least amount of time" on. Therefore, it is hard to apply this generalization directly when both overall running time and overall accuracy need to be considered. On the other hand, it is worth noting that their argument does apply to our Group Range Problem if only zero-error data structures are considered.

1.2 Further Results

The fact that Theorem 1.1 holds for any group G, in addition to more 'common' groups like $\mathbb{Z}_m$ and permutation groups, makes it quite versatile. For example, it holds when G is the direct product of many smaller groups. In this case, the problem can be viewed as many disjoint copies of the Group Range problem on the smaller component groups with simultaneous updates, yet the lower bound remains.

The case where G is the general linear group of invertible matrices also has many applications; see Appendix A for a discussion of applications to physics and to other dynamic data structure problems. In this case, we show how the matrix structure can be exploited to prove further results. We consider a variant of the original problem, which we call the *Matrix Product Problem*, where queries can only ask for a bit about the bottom-right entry of the product of the *entire* range of matrices, rather than any bit about the product of any subrange, and in Section 4 we show that the lower bound still applies:

Corollary 1.3. *Theorem 1.1 holds for the Matrix Product Problem.*

We show a similar result for upper-triangular matrices as well in Section 4.1.

It would be interesting to extend Theorem 1.1 to hold for an even wider class of algebraic structures. For instance, some past work (e.g. [PD04]) considers the partial sums problem where G is any *semi-group*. We show that such an extension is impossible, not only to semi-groups, but even for monoids (a type of algebraic structure between groups and semi-groups, which satisfies all the group axioms except the existence of inverses). Indeed, we show in Section 3.2 that the $\Omega(\log n)$ lower bound can be beaten for the *Monoid Range Problem* (the same as the Group Range Problem except that G can be any monoid), so no general lower bound applies:

Theorem 1.4. *There exists a family of monoids $(G_n)_n$ such that the Monoid Range Problem can be solved in $O\left(\frac{\log n}{\log\log n}\right)$ time per operation worst-case deterministically in the cell-probe model.*

1.3 Organization

We first formally define the online communication model in Section 2.1, and then we prove the online set intersection lower bound in Section 2.2. In Section 3.1, we prove our main result, the cell-probe lower bound for the Group Range Problem. In Section 3.2, we show that the invertibility of group elements is necessary for the lower bound. In Section 4, we prove stronger results for the special case of the group G of invertible matrices.

2 The Online Communication Model

In this section, we define the online communication model, and then we present a general approach for proving lower bounds in it. As a demonstration of our approach, we prove a concrete (and almost tight) lower bound for the online set-intersection problem.

2.1 Model Definition

In the online communication model, two players Alice and Bob are given inputs $X \in \mathcal{X}$ and $Y_1, Y_2, \ldots, Y_k \in \mathcal{Y}$ *gradually*. Alice and Bob want to compute $f(X, i, Y_i)$ for every $i \in [k]$. Alice receives X at the beginning, while the Y_i's are revealed to the players gradually as follows.

1. The game consists of k stages. The players may communicate at any time as if they were in the classic communication model. The players remember the transcript from previous stages.
2. At the beginning of Stage i for $i \in [k]$, Y_i is revealed to *Bob*.
3. Next, the players communicate so that Bob can compute $f(X, i, Y_i)$.
4. At the end of Stage i, Y_i is revealed to *Alice*, and the players proceed to the next stage.

In a deterministic (resp. randomized) online communication protocol, the players communicate as if they were in the deterministic (resp. randomized) communication model in the second step of each stage.

2.2 Online Set-Intersection Lower Bound

Online Set-Intersection. In the online set-intersection problem (OSI), Alice is given one set X of size k over the universe $[n]$. In each stage, Bob is given an input $Y_i \in [n]$, which is an element in the same universe. The goal of this stage is to verify whether $Y_i \in X$. Equivalently, the inputs are two (multi-)sets $X, Y \subseteq [n]$ of size k each. Each element of the set Y is revealed one by one. The goal is to compute their intersection.

Theorem 2.1. *For $n \geq k^2$, any zero-error OSI protocol using public randomness must have expected total communication cost at least $\Omega(k \log \log k)$.*

As a complement to our lower bound, the following simple protocol has expected communication cost $O(k \log \log n + |X \cap Y| \cdot \log n)$:

1. The players use public randomness to sample a hash function $h : [n] \to [k \log n]$.
2. Alice sends Bob the set $h(X)$ in $O(\log \binom{k \log n}{k}) = O(k \log \log n)$ bits.
3. For each Y_i, if $h(Y_i)$ is not in $h(X)$, Bob returns "NO" immediately. Otherwise, Bob solves this stage trivially by sending Alice Y_i.

For each $Y_i \notin X$, the probability that $h(Y_i) \in h(X)$ is at most $1/\log n$. Therefore, the total expected communication cost for stages such that $Y_i \notin X$ is $O(k)$. Thus, the above protocol has the claimed total communication cost. Our lower bound is tight as long as $n = 2^{\log^{O(1)} k}$ and $|X \cap Y|$ is not too large.

In the following, we prove the communication lower bound. First by Yao's Minimax Principle [Yao77], we may fix an input distribution and assume the protocol is deterministic. Now let us consider the following hard distribution.

Hard distribution. We take the first k^2 elements from the universe, and divide them into k blocks of size k each. X will contain one uniformly random element from each block independently. Each Y_i will be a uniformly random element from the first k^2 elements. Different Y_i's are chosen independently.

The high-level idea of the proof is to first reduce from OSI to a classic (non-online) communication complexity problem. In particular, we consider the problem solved in each stage of the OSI problem: Alice is given a set of k elements from a universe of size n and Bob is given a single element from the same universe, and their goal is to determine if Bob's element is in Alice's set. This is precisely the *index* problem. Then we focus on the stage that costs the least amount of communication, and show an index lower bound with respect to this stage. The hard distribution for OSI induces the following hard distribution for index.

Hard distribution for index. Divide the first k^2 elements of the universe into k blocks of size k each. Alice's set X consists of one uniformly random elements from each block independently. Bob's element y is chosen from the first k^2 elements uniformly at random.

The following lemma characterizes OSI protocols by (classic) index protocols.

Lemma 2.2. *Under the above hard input distributions, there is a protocol for OSI which uses $O(g(n,k))$ bits in expectation if and only if there is a protocol for index where Alice first sends a message of expected length of $O(g(n,k))$ bits and then Alice and Bob only speak an additional $O(g(n,k)/k)$ bits in expectation.*

Proof. We first prove the forward direction, which is more nuanced. Suppose we have a protocol P for OSI, and we want such a protocol P' for index. The main idea is to fix the stage in which the players send the least bits in expectation, and embed the index problem into this stage of OSI. Let the i-th stage be one stage in which the players only speak $O(g(n,k)/k)$ bits in expectation. To solve the index problem on inputs X and y, we invoke the following protocol P':

1. The players use public randomness to sample $Y_1, Y_2, \ldots, Y_{i-1}$ according to the hard distribution for OSI;
2. Now Alice has all the information for the first $i-1$ stages, she simulates P for *both* players, and sends Bob the entire transcript;
3. The players pretend that they in the i-th stage of OSI with $Y_i = y$, and continue to simulate P from the transcript Alice sent in the previous step;
4. Bob outputs $y \in X$ if and only if the output of P indicates that $Y_i \in X$.

In this protocol P', the first message is sent by Alice, which has expected length no more than that of the transcript of P, which is $O(g(n,k))$. Then the players simulates the i-th stage of P. Note that the distribution of first i stages of the simulate is identical to the first i stages of the hard distribution. Thus, the expected communication cost of step 3 is at most $O(g(n,k)/k)$. Since the goal is stage i is to determine if $Y_i \in X$, which is precisely if $y \in X$ by the way we set up the simulation, the output of P' will be correct if P is correct.

We finish with the easy reverse direction. Suppose there is such a protocol P' for index; we want to construct a protocol P for OSI. We begin by noticing that Alice's first $O(g(n,k))$ bits can only depend on her input. We begin by having Alice simulate P' and sends the long message before Bob begins speaking. Now in each stage, for Bob's input Y_i, we have Alice and Bob simulate P' on (X, Y_i), but skipping the first $O(g(n,k))$ bits since they have already been communicated.

In this protocol, Alice sends $O(g(n,k))$ bits in her first message. Then in each stage, only $O(g(n,k)/k)$ bits are transmitted between the players. Thus, the total communication cost is $O(g(n,k))$ in expectation. □

Let P' be a zero-error protocol for index such that Alice first sends c_0 bits in expectation, and then Alice and Bob communicate for c_A and c_B bits respectively (in expectation). The following lemma lower bounds c_0, c_A, c_B.

Lemma 2.3. *For sufficiently large k, any such P' must have either*

- $c_0 \geq \frac{1}{7} k \log k$, *or*
- $c_A \geq c_0 \cdot 2^{-13 \max\{c_B, 1\} \cdot 2^{6c_0/k}}$.

The main idea of the proof is to let Alice simulate Bob. For simplicity, let us first assume the protocol has three rounds: Alice sends c_0 bits, then Bob sends c_B bits, finally Alice sends c_A bits. To simulate Bob, Alice goes over all possible messages that Bob could send, then for each message, sends Bob what she would say if she received that message. If Bob sends at most c_B bits *in worst case*, Alice will be able to complete the above simulate in $c_0 + c_A \cdot 2^{c_B}$ bits of communication. Then Bob will be output whether his input Y_i is in Alice's set X. In particular, Alice's message depends only her input X, and Bob can do so for any Y_i. That is, Bob will be able to recover the set X based only on this message, which yields a lower bound on c_0, c_A, c_B.

Proof of Lemma 2.3. Without loss of generality, we may first assume $c_B \geq 1$. By Markov's inequality and a union bound, for any $C \geq 2$, with probability at least $1 - 2/C$, Alice sends no more than $C \cdot c_A$ bits and Bob sends no more than $C \cdot c_B$ bits after Alice's first message. The next step is to let Alice *simulate* the entire protocol, and turn it into *one-way communication*.

More specifically, the transcript π of a conversation between Alice and Bob is a binary string, in which each bit represents the message sent in the chronological order. Given π and a fixed protocol, there shall be no ambiguity in which bits are sent by which player. That is, for any π, we can always decompose it into (π_A, π_B), where π_A is a binary string obtained by concatenating the bits sent by Alice in the chronological order, and similar for π_B. On the other hand, given (π_A, π_B), there is a *unique* way to combine them into a single transcript π, since a prefix of the transcript uniquely determines the player who speaks the next. We know that with probability at least $1 - 2/C$, $|\pi_A| \leq C \cdot c_A$ and $|\pi_B| \leq C \cdot c_B$. In the new protocol, after Alice sends the first $c_0 \cdot k$ bits, she goes over all $2^{C \cdot c_B}$ strings s of length at most $C \cdot c_B$. For each s (in alphabetical order), she sends the first $C \cdot c_A$ bits of π_A based on her input *assuming* $\pi_B = s$. That is, Alice tells Bob that "if s was your first $C \cdot c_B$ bits of the conversation, then here is what I would say for my first $C \cdot c_A$ bits." In total, she sends another $C \cdot c_A \cdot 2^{C \cdot c_B}$ bits. Thus, Bob can figure out the answer based only on the above messages, with probability $1 - 2/C$ (over the random input pairs). To balance the lengths of two messages, we set $C = \frac{1}{2c_B} \log \frac{c_0}{c_A}$. If $C < 2$, then we have $\log \frac{c_0}{c_A} < 4c_B$, and thus

$$c_A > c_0 \cdot 2^{-4c_B},$$

which implies the second inequality in the statement. Otherwise, the above argument holds, and we have

$$\begin{aligned} C \cdot c_A \cdot 2^{C \cdot c_B} &= C \cdot c_A \cdot \sqrt{\frac{c_0}{c_A}} \\ &= \frac{c_A}{2c_B} \cdot \left(\sqrt{\frac{c_0}{c_A}} \log \frac{c_0}{c_A} \right) \\ &\leq c_A \cdot \left(\sqrt{\frac{c_0}{c_A}} \log \sqrt{\frac{c_0}{c_A}} \right) \\ &\leq c_A \cdot \frac{c_0}{c_A} = c_0. \end{aligned}$$

Thus, Alice sends at most $2c_0$ bits in expectation in total. This message only depends on her input X. By Markov's inequality, for at least $2/3$ of the X's, Alice sends no more than $6c_0$ bits. By Markov's inequality again, for at least $2/3$ of the X's, the probability (over a random y) that Bob can figure out if $b \in A$ based only on Alice's first message is at least $6/C$. Since there are k^k different possible X's, at least $k^k/3$ different X's have both conditions hold. Thus, there must be $k^k/3 \cdot 2^{-6c_0}$ such X's that Alice sends the same message M. Denote this set of X's by $\mathcal{X}$. Moreover, when M is the message Bob receives, there are at least $(1 - 6/C)k^2$ different y's such that Bob can figure out the answer based only on the value of y and M. Denote this set of y's by $\mathcal{Y}$. In the combinatorial rectangle $\mathcal{R} = \mathcal{X} \times \mathcal{Y}$, for every $y \in \mathcal{Y}$, either $y \in X$ for every $X \in \mathcal{X}$, or $y \notin X$ for every $X \in \mathcal{X}$. That is, $\mathcal{R}$ is a *column-monochromatic rectangle*[4] of

[4] A rectangle with the same function value in every column.

size $(k^k/3 \cdot 2^{-6c_0}) \times (1-6/C)k^2$.

On the other hand, for the index problem, in any column-monochromatic rectangle $\mathcal{R} = \mathcal{X} \times \mathcal{Y}$, the answer is "YES" in no more than k columns of $\mathcal{Y}$ (the element is in the set). This is because each set $X \in \mathcal{X}$ has size k. In order to upper bound the number of $y \in \mathcal{Y}$ that is not in any X, let r_i for $1 \le i \le k$ be the size of the intersection of $\mathcal{Y}$ and the i-th block of the universe. Thus, the number of X's that avoids all $y \in \mathcal{Y}$ is at most

$$(k-r_1)(k-r_2)\cdots(k-r_k) \le \left(k - \frac{1}{k}(r_1+\cdots+r_k)\right)^k$$

by the AM-GM inequality. That is, at most $k^2 - k|\mathcal{X}|^{1/k}$ y are not in any X. Overall, we have $|Y| \le k + k^2 - k|\mathcal{X}|^{1/k}$. Combining this with the parameters from the last paragraph, we get

$$(1-6/C)k^2 \le k + k^2 - k\left((k^k/3\cdot 2^{-6c_0})\right)^{1/k}.$$

Simplifying the inequality yields

$$6/C \ge 2^{-6c_0/k}\cdot 3^{-1/k} - 1/k.$$

When $c_0 < \frac{1}{7}k\log k$, we have $2^{-6c_0/k}\cdot 3^{-1/k} - 1/k > \frac{12}{13}\cdot 2^{-6c_0/k}$ for sufficiently large k. Pluging-in the value of $C(=\frac{1}{2c_B}\log\frac{c_0}{c_A})$ and simplifying, we obtain

$$c_A \ge c_0 \cdot 2^{-13c_B/2^{-6c_0/k}}.$$

This proves the lemma. □

Proof of Theorem 2.1. For any OSI protocol with total communication cost c, by Lemma 2.2 and Lemma 2.3, we have either

- $c \ge \frac{1}{7}k\log k$, or
- $c/k \ge c\cdot 2^{-13\max\{c/k,1\}\cdot 2^{6c/k}}$.

The second inequality simplifies to $\max\{c/k,1\}\cdot 2^{\Theta(c/k)} \ge \Omega(\log k)$. Thus, we must have $c \ge \Omega(k\log\log k)$. □

3 The Group Range Problem

3.1 Lower Bound Proof

The goal of this subsection is to prove our main result:

Theorem 1.1 (restated). *There exists a distribution over n updates and queries for the Group Range Problem with binary encoding of the group elements $e: G \to \{0,1\}^s$, such that for any randomized cell-probe data structure D with word size $w = \Theta(\log n)$, which with probability p answers at least a $(\frac{1}{2}+\delta)$ fraction of queries correctly and spends $\epsilon n\log n$ total running time, we must have $p \le \exp(-\delta^2 n)$, as long as $s \le (1+\epsilon)\log|G|$, $\delta^2 \gg \epsilon \ge \Omega(1/\log n)$, and n is sufficiently large.*

For convenience, we will assume that n is a power of two. A similar argument applies to the general case. We will also say that the data structure succeeds on an input when the event described occurs: it answers a $(\frac{1}{2}+\delta)$ fraction of queries correctly and spends at most $\epsilon n\log n$ total running time.

Our proof is divided into three steps. First, we construct a random input sequence so that we can apply Yao's minimax principle and consider a determinstic data structure. Second, we consider various

Shorthand	Operation
u_0	Update$(0, \mathcal{U}_G)$
q_0	Query$(0, \mathcal{U}_{[n]}, \mathcal{U}_{[s]})$
u_1	Update$(2, \mathcal{U}_G)$
q_1	Query$(0, \mathcal{U}_{[n]}, \mathcal{U}_{[s]})$
u_2	Update$(1, \mathcal{U}_G)$
q_2	Query$(0, \mathcal{U}_{[n]}, \mathcal{U}_{[s]})$
u_3	Update$(3, \mathcal{U}_G)$
q_4	Query$(0, \mathcal{U}_{[n]}, \mathcal{U}_{[s]})$

Figure 1: Structure of our random input sequence ($n = 4$). Here, $\mathcal{U}_S$ is an entry drawn from the uniform distribution on set S.

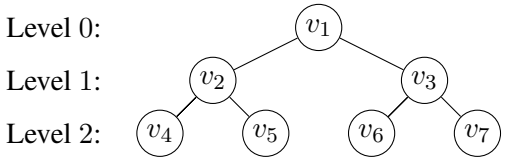

Figure 2: Our subproblems correspond to nodes of a balanced binary tree ($n = 4$).

subproblems of this sequence. We show that the data structure must do well on at least one of them, but with some additional structure on how it probes cells when solving this subproblem. Third, we use the data structure on this subproblem to produce a communication protocol for a problem in our online communication complexity model.

3.1.1 Step One: The Hard Distribution

Our random input sequence for D is as follows. Our sequence $u_0, q_0, u_1, q_1, \ldots, u_{n-1}, q_{n-1}$ consists of $2n$ alternating update and query operations. Intuitively, we want the update operations in any contiguous window to look relatively spaced out, and we want query operations to check a random interval that starts at the beginning of the sequence.

More formally, let $rev(i)$ be the integer whose s-bit binary representation is the reverse of the representation of i. For example, $rev(0) = 0$ and $rev(1) = 2^{n-1}$. Update operation u_i will set the $rev(i)^{th}$ group element to a uniform random group element. Query operation q_i will use the range $[0, R]$ where R is drawn uniformly from $\{0, 1, \ldots, n-1\}$ and ask for a random bit of that group product. Figure 1 shows what a random input looks like in the $n = 4$ case.

Yao's minimax principle guarantees that since D is a randomized structure with guarantees on worst-case inputs, there must be a deterministic data structure D' with the same guarantees on a random input sequence of this form.

3.1.2 Step Two: Identifying Efficient Subproblems

Our next step is to divide an input from this distribution into several subproblems. Since D' should do well on the entire input, it must also do well on many subproblems. We will then be able to use the fact that D' does well on a subproblem to derive an efficient communication protocol for a related communication game, in the next step of the proof.

In each subproblem, we consider two equally-sized adjacent intervals of operations, I_A (Alice's interval) and I_B (Bob's interval). Each subproblem corresponds to the node of a balanced binary tree with $\log_2 2n$

Operations	u_0	q_0	u_1	q_1	u_2	q_2	u_3	q_3
Level 0	$I_A(v_1)$				$I_B(v_1)$			
Level 1	$I_A(v_2)$		$I_B(v_2)$		$I_A(v_3)$		$I_B(v_3)$	
Level 2	$I_A(v_4)$	$I_B(v_4)$	$I_A(v_5)$	$I_B(v_5)$	$I_A(v_6)$	$I_B(v_6)$	$I_A(v_7)$	$I_B(v_7)$

Figure 3: Division into subproblem intervals ($n = 4$).

levels. Such a tree has $(2n-1)$ nodes: $v_1, \ldots, v_{2n-1}$, where v_1 is the root node and the children of v_i are v_{2i} (left) and v_{2i+1} (right). We denote node v's corresponding intervals as $I_A(v)$ and $I_B(v)$.

Level 0 of our tree consists of just the root node v_1. Its corresponding subproblem concerns interval $I_A(v_1)$ as the first half of the entire sequence, and $I_B(v_1)$ as the second half of the entire sequence. The other subproblems are defined recursively as follows. If v_i is a left child, then $I_A(v_i)$ is the left half of $I_A(v_{i/2})$ and $I_B(v_i)$ is the right half. If v_i is a right child, then $I_A(v_i)$ is the left half of $I_B(v_{(i-1)/2})$ and $I_B(v_i)$ is the right half. See Figure 3 for the intervals of the $n = 4$ case.

Definition 1. *The set of nodes in level j is denoted $\ell(j)$ and consists of $\{v_{2^j}, \ldots, v_{2^{j+1}-1}\}$. The set of cells that the data structure probes when processing the operations of $I_A(v)$ is $P_A(v)$ (P stands for probes), and similarly for $I_B(v)$ and $P_B(v)$.*

We are now ready to connect the running time of the data structure with a property of these subproblems. Consider the sum $\sum_{j \in [\log 2n]} \sum_{v \in \ell(j)} |P_A(v) \cap P_B(v)|$. Each time the data structure probes a cell, it contributes to this summation in at most one subproblem v: the one where its previous access to the cell was in $P_A(v)$ and its current access is in $P_B(v)$. When the data structure succeeds on an input, this summation is is at most $2\epsilon n \log(2n)$, since it is at most the running time of D' on $2n$ operations.

We know that overall the data structure succeeds on a $(\frac{1}{2} + \delta)$ fraction of queries, and that the sum of these intersection sizes is small. We wish to identify subproblems v with similar guarantees, namely: (i) $|P_A(v) \cap P_B(v)|$ is small (on the order of $\epsilon |P_A(v)|$) and (ii) D' answers many queries in $I_B(v)$ correctly. In step three of the proof, we will show that such a subproblem must have a low probability of success. In particular, we will prove the following lemma in step three, and the remainder of this step is concerned with identifying subproblems with these guarantees as efficiently as possible:

Lemma 3.1. *For two intervals $I_A = I_A(v)$ and $I_B = I_B(v)$ consisting of k updates and queries each. Then the probability that*

- $|P_A(v) \cap P_B(v)| \leq \epsilon_v \cdot k$ *and*
- *D answers a $(1/2 + \delta_v)$-fraction of queries in $I_B(v)$ correctly*

conditioned on all operations O before $I_A(v)$ is at most

$$\exp(-(\delta_v - c \cdot (\sqrt{\epsilon} + \sqrt{\epsilon_v}))^2 \cdot k)$$

for some constant c, as long as $k \gg \log n$ and $\delta_v - c \cdot (\sqrt{\epsilon} + \sqrt{\epsilon_v}) \geq 0$.

Using Lemma 3.1, we can now prove Theorem 1.1. The main idea to find our desired subproblem is to find a level that is both "efficient" ($\sum_v |P_A(v) \cap P_B(v)|$ is small) and "accurate" (a decent fraction of the queries in $I_B(v)$'s are correct). Since we only care about the queries in $I_B(v)$'s, we first argue that as long as $(1/2 + \delta)$-fraction of the queries are correct overall, then the sum of correctly answered queries over all I_B's in all levels cannot be small. To find such a desired level, we observe from the above lemma that the only important parameter is $\delta_v - c \cdot (\sqrt{\epsilon} + \sqrt{\epsilon_v})$. We then apply an averaging argument to conclude the existence of a level with a large average value of $\delta_v - c \cdot (\sqrt{\epsilon} + \sqrt{\epsilon_v})$. In the following, we formally prove the theorem.

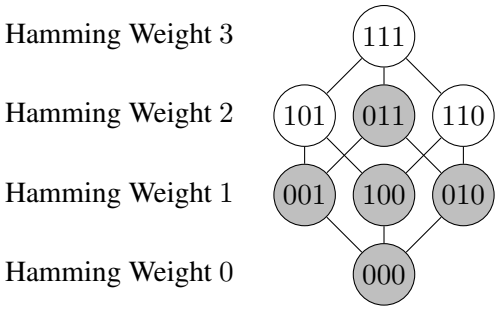

Figure 4: Queries correspond to vertices of a hypercube ($n = 8$). In this example, $\frac{5}{8} = 62.5\%$ of queries in the big hypercube are shaded, but the average small subhypercube H_i only has $\frac{1+2+2}{12} = 41.\bar{6}\%$ of its queries shaded, and no small hypercube H_i has more than $\frac{2}{4} = 50\%$ of its queries shaded.

Proof of Theorem 1.1. Our plan is to apply Lemma 3.1 to all the subproblems of an entire level. We want such a level to require little communication, yet also gets many queries correct. Additionally, since we require that $k \gg \log n$, this level cannot be near the bottom of the tree. The following definitions will help us perform this analysis on levels:

Definition 2. *Let δ_j be a random variable so that a $(\frac{1}{2} + \delta_j)$ fraction of the queries in $\cup_{v \in \ell(j)} I_B(v)$ are answered correctly. Let ϵ_j be a random variable so that $\sum_{v \in \ell(j)} |P_A(v) \cap P_B(v)| = \epsilon_j n$.*

It seems intuitive that if the data structure answers many queries correctly, it must also do so over all intervals $I_B(v)$. Unfortunately, this claim is not cut and dried; some queries may show up a logarithmic number of times in intervals $I_B(v)$, while others only show up once. For example, in Figure 3, q_0 shows up in one subproblem, q_1 and q_2 show up in two, and q_3 shows up in three. However, we can say something about which levels q_i shows up in some $I_B(v)$. Ignoring the final level of the tree (we can't use the bottom levels anyway), if we write i as a $(\log n)$-bit binary number, then it shows up in level j exactly when its j^{th} bit (starting from most-significant) is a one. Returning to our Figure 3 example, q_1 can be written as q_{01_2} and hence does not show up in level 0 but does show up in level 1, while q_3 can be written as q_{11_2} and hence shows up in both level 0 and 1.

Let $d = \log n$; we can visualize our queries as vertices of the d-dimensional hypercube H. As discussed above, the $I_B(v)$'s of a level correspond to the $(d-1)$-dimensional subhypercubes $H_i = \{x \in H \mid x_i = 1\}$. If we have a 0-1 labelling of the big hypercube with a $(\frac{1}{2} + \delta)$ fraction labelled 1, let us consider the smallest average fraction of the subhypercubes needs to be labeled 1. Figure 4 demonstrates that the fraction may decrease. We aim to show that this drop cannot be that large.

Since a node of Hamming weight j contributes to exactly j subhypercubes, the worst case labelling occurs when nodes with the lowest Hamming weight are labeled 1 first. Suppose $(1/2 + \delta)$-fraction of nodes have Hamming weight at most l, i.e., let l be the largest integer such that

$$\sum_{j=0}^{l} \binom{d}{j} \leq (1/2 + \delta)2^d.$$

Therefore, we have $\sum_{j=\lceil d/2 \rceil + 1}^{l} \binom{d}{j} \geq (\delta - O(1/\sqrt{d}))2^d$. Then the average fraction of labelled nodes in

the subhypercubes is at least

$$\begin{aligned}
\frac{1}{d}\sum_{j=0}^{l}\binom{d}{j}\cdot j\cdot 2^{-(d-1)} &= \frac{1}{d}\sum_{j=0}^{\lceil d/2\rceil}\binom{d}{j}\cdot j\cdot 2^{-(d-1)} + \frac{1}{d}\sum_{j=\lceil d/2\rceil+1}^{l}\binom{d}{j}\cdot j\cdot 2^{-(d-1)} \\
&\geq \sum_{j=1}^{\lceil d/2\rceil}\binom{d-1}{j-1}\cdot 2^{-(d-1)} + \frac{1}{d}\sum_{j=\lceil d/2\rceil+1}^{l}\binom{d}{j}\cdot \frac{d}{2}\cdot 2^{-(d-1)} \\
&\geq \frac{1}{2} + \sum_{j=\lceil d/2\rceil+1}^{l}\binom{d}{j}\cdot 2^{-d} \\
&\geq \frac{1}{2} + \delta - O(1/\sqrt{\log n}).
\end{aligned}$$

Translating back to our original problem, we have shown that $\text{avg}_{i\in[\log n]}\delta_i = \delta - O(\frac{1}{\sqrt{\log n}})$. However, we would like to avoid the bottom levels of the tree, since Lemma 3.1 requires k to be bigger than $\log n$. Hence we restrict to the top $L = \frac{1}{3}\log n$ levels of our tree, where $k \geq n^{2/3}$. We would like to show that $\text{avg}_{i\in[L]}\delta_i = \delta - O(\frac{1}{\sqrt{\log n}})$ as well. Fortunately, the same proof works even when we ignore a constant fraction of coordinates. Each point now has duplicity $n^{2/3}$, and we still end up with the fact that an average subhypercube has a $\left(\frac{1}{2} + \delta - O(\frac{1}{\sqrt{\log n}})\right)$ fraction of 1's, albeit with the dimension now L instead of $\log n$.

This brings us halfway to our objective of identifying a level j with good guarantees on δ_j and ϵ_j. We next claim that if the data structure succeeds, there is some level $j \in [L]$ such that $(\delta_j - c(\sqrt{\epsilon} + \sqrt{\epsilon_j}))^2$ is large.

We know that $\text{avg}_{j\in[L]}\delta_j = \delta - O\left(\frac{1}{\sqrt{\log n}}\right)$. Next, we know that $\text{avg}_{j\in[L]}\epsilon_j \leq 3\epsilon n$, since the average of one-third of all levels cannot be larger than three times the overall average (this is analagous to Markov's inequality). Hence by Jensen's inequality, we know that:

$$\sum_{j\in[L]}(\delta_j - c(\sqrt{\epsilon} + \sqrt{\epsilon_j})) \geq L\left(\delta - O\left(\frac{1}{\sqrt{\log n}}\right) - 2c\sqrt{3\epsilon}\right)$$

Thus,

$$\begin{aligned}
\max_{j\in[L]}(\delta_j - c(\sqrt{\epsilon} + \sqrt{\epsilon_j})) &\geq \delta - O\left(\frac{1}{\sqrt{\log n}}\right) - 2c\sqrt{3\epsilon} \\
&\geq \Omega(\delta).
\end{aligned}$$

Hence whenever our algorithm succeeds, there must be some level $j \in [L]$ with this guarantee on δ_j and ϵ_j. However, the level j may differ depending on the random input that the algorithm succeeds on. We do know that there must be some level $j \in [L]$ which has this guarantee with at least $\frac{p}{L}$ probability. It will suffice to prove that $\frac{p}{L}$ is exponentially small, since L is absorbed into our $\exp$ notation.

We have now identified our level of interest, and we want to apply Lemma 3.1 to its subproblems. Analagous to how we defined communication parameters ϵ_j and success parameters δ_j for levels, we can define these parameters for each subproblem:

Definition 3. *Let δ_v be a random variable so that a $(\frac{1}{2} + \delta_v)$ fraction of the queries in $I_B(v)$ are answered correctly. Let ϵ_v be a random variable so that $|P_A(v) \cap P_B(v)| = \epsilon_v k$ where $k = |I_A(v)| = |I_B(v)|$.*

Fix sequences $\{\epsilon_v\}_{v\in\ell(j)}$ and $\{\delta_v\}_{v\in\ell(j)}$ such that $\sum_{v\in\ell(j)}\epsilon_v = \epsilon_j n/2k$ and $\sum_{v\in\ell(j)}\delta_v = \delta_j n/2k$. We apply Lemma 3.1 to each subproblem $v \in \ell(j)$ with parameters ϵ_v and δ_v such that $\delta_v - c\cdot(\sqrt{\epsilon_v} + \sqrt{\epsilon}) \geq 0$.

By noticing that all the interval pairs $(I_A(v), I_B(v))$ are disjoint, using Jensen's inequality, and the fact that $\delta_j - O(\sqrt{\epsilon_j} + \sqrt{\epsilon}) > 0$, we have the probability that:

- $\forall v \in \ell(j)$, $|P_A \cap P_B| \leq \epsilon_v \cdot k$ and
- $\forall v \in \ell(j)$, D answers a $(1/2 + \delta_v)$-fraction of queries in $I_B(v)$ correctly

is at most

$$\begin{aligned}
\prod_{v \in \ell(j)} \exp(-(\max\{0, \delta_v - c \cdot (\sqrt{\epsilon} + \sqrt{\epsilon_v})\})^2 \cdot k) &= \exp\left(- \sum_{v \in \ell(j)} (\max\{0, \delta_v - c \cdot (\sqrt{\epsilon} + \sqrt{\epsilon_v})\})^2 \cdot k \right) \\
&\leq \exp\left(- \left(\frac{k}{n} \cdot \sum_{v \in \ell(j)} \max\{0, \delta_v - c \cdot (\sqrt{\epsilon} + \sqrt{\epsilon_v})\} \right)^2 \cdot n \right) \\
&\leq \exp\left(- \left(\frac{k}{n} \cdot \sum_{v \in \ell(j)} (\delta_v - c \cdot (\sqrt{\epsilon} + \sqrt{\epsilon_v})) \right)^2 \cdot n \right) \\
&\leq \exp\left(- \left(\delta_j - O(\sqrt{\epsilon_j} + \sqrt{\epsilon}) \right)^2 \cdot n \right).
\end{aligned}$$

We can finish applying a union bound over all such sequences $\{\epsilon_v\}_{v \in \ell(j)}$ and $\{\delta_v\}_{v \in \ell(j)}$. There are $\binom{\epsilon_j n + n/k}{n/2k}$ many $\{\epsilon_v\}$'s and $\binom{O(n)}{n/2k}$ many $\{\delta_v\}$'s. The probability of level j having good guarantees can be at most:

$$\begin{aligned}
&\binom{\epsilon_j n + n/k}{n/k} \cdot \binom{O(n)}{n/k} \cdot \exp\left(- \left(\delta_j - O(\sqrt{\epsilon_j} + \sqrt{\epsilon}) \right)^2 \cdot n \right) \\
&\leq \exp(n/k \cdot \log k) \cdot \exp\left(- \left(\delta_j - O(\sqrt{\epsilon_j} + \sqrt{\epsilon}) \right)^2 \cdot n \right).
\end{aligned}$$

Since $\frac{1}{k} \log k \ll (\delta_j - O(\sqrt{\epsilon_j} + \sqrt{\epsilon}))^2$ and we picked j so that $(\delta_j - O(\sqrt{\epsilon_j} + \sqrt{\epsilon}))^2 > \delta^2$, this yields our desired result.

□

3.1.3 Step Three: The Communication Game

The main goal of this step is to prove Lemma 3.1. The key idea is to show how an efficient data structure can be used to produce an efficient communication protocol for a particular communication game, and then to rule out the possibility of an efficient communication protocol, hence proving that the original efficient data structure could not exist. We begin by defining the communication game on interval pairs we will be focusing on, which uses our online communication model from Section 2.1.

Communication Game We define one communication game for each interval pair (I_A, I_B). Fix two intervals $I_A = I_A(v)$ and $I_B = I_B(v)$ consisting of k updates and queries each, all operations O prior to these intervals, all queries Q_A in I_A and all updates U_B in I_B. That is, the only undetermined operations up to the end of I_B are the updates in I_A and the queries in I_B. We embed these operations into a communication game. In the associated online communication game $G = G(v, O, Q_A, U_B)$, X consists of the updates in I_A, and Y_i is the i^{th} query in I_B. The goal of Stage i is to compute the i^{th} query in I_B.[5]

[5]Note that the previous queries do not affect the output of the i^{th} query.

Input Distribution The input X is sampled as a random set of updates in $I_A(v)$ and $(Y_1, \ldots, Y_k)$ as a random set of queries in $I_B(v)$ under our hard distribution for the Group Range problem.

Lemma 3.2. *Consider two intervals $I_A(v)$ and $I_B(v)$, consisting of k updates and queries each. Let the operations prior to them be O, the queries in $I_A(v)$ be Q_A, the updates in $I_B(v)$ be U_B. For any data structure D for the group range problem and ϵ_v, there is a protocol $\mathcal{P}_D$ for the communication game $G(v, O, Q_A, U_B)$ such that*

1. *Alice sends $O(\epsilon_v \cdot k \cdot \log n)$ bits;*
2. *Bob sends no message;*
3. *For every δ_v, the probability that $\mathcal{P}_D$ answers $(1/2 + \delta_v - \epsilon_v)$-fraction of the $f(X, i, Y_i)$'s correctly is at least*

$$\Pr\left[|P_A \cap P_B| \leq \epsilon_v \cdot k, D \text{ answers a } (1/2 + \delta_v)\text{-fraction of queries in } I_B \text{ correctly} \mid O, Q_A, U_B\right].$$

Lemma 3.3. *For any protocol P for $G(v, O, Q_A, U_B)$ and ϵ_v, δ_v where:*

1. *Alice sends $O(\epsilon_v k \cdot \log n)$ bits, and*
2. *Bob sends no message*

must have

$$\Pr[P \text{ answers } (1/2 + \delta_v - \epsilon_v)\text{-fraction of the } f(X, i, Y_i)\text{'s correctly}] \leq \exp(-(\delta_v - O(\sqrt{\epsilon} + \sqrt{\epsilon_v}))^2 \cdot k).$$

Lemma 3.1 follows directly from applying both Lemma 3.2 and Lemma 3.3. Hence it remains to prove these two lemmas.

Proof of Lemma 3.2. The idea is that the players simulate D as operations are revealed, and Alice sends some necessary information to Bob. Consider the following protocol $\mathcal{P}_D$:

1. (Preprocessing) Recall that Alice knows all operations up to the end of I_A and the updates in I_B, Bob knows all operations prior to I_A and all operations in I_B. First, Alice simulates D up to the end of I_A, and Bob simulates D up to the beginning of I_A and *skips* I_A. Denote the memory state that Alice has *at this moment* by M_A. Next, the players are going to simulate operations in I_B.
2. (Stage i - Alice's simulation) Since the $(i-1)$-th query is revealed to Alice in the last stage, Alice continues the simulation up to the i-th query. Alice sends Bob the cells (their addresses and contents in M_A) that are
   - probed during this part of the simulation, and
   - probed during I_A, and
   - not probed in the previous stages.

   That is, Alice sends Bob all cells in $P_A \cap P_B$ that are just probed for the very first time among all stages so far.
3. (Stage i - Bob's simulation) Bob first updates his memory state according to Alice's message: For each cell in the message, Bob replaces its content with the actual content in M_A. Since this is the first time D probes these cells, their contents remain the same as in M_A. Bob then continues the simulation up to i-th query.

4. (Stage i - query answering) Bob simulates D on query Y_i. During the simulation, Bob *pretends* that he has the right memory state for the query, even though he has skipped I_A, and only has received partial information about it. Then he outputs the same answer as D does. Finally, Bob rolls back the memory to the version right before this query (after simulation described in Step 3). That is, since the simulation on this query may be incorrect, Bob does not make any real changes to the memory in this step.

5. As soon as Alice has sent $2\epsilon_v k \cdot w + 1$ bits (where w is the word-size), the players stop following the above steps, and output uniform random bit for all queries from this point.

Analyzing the Protocol It is easy to verify that Bob sends no message, and due to the last step, Alice always sends no more than $O(\epsilon_v k \cdot \log n)$ bits (word-size $w = \Theta(\log n)$). Thus, $\mathcal{P}_D$ has the first two properties claimed in the lemma statement. In following, we are going to show that whenever $|P_A \cap P_B| \leq \epsilon_v k$ and $(1/2 + \delta_v)$-fraction of queries in I_B are correct, $\mathcal{P}_D$ answers at least $(1/2 + \delta_v - \epsilon_v)$-fraction of the queries correctly, which implies the third property.

In Step 2, Alice only sends Bob cells in $P_A \cap P_B$. Moreover, each cell in the intersection will only be sent once - in the stage when it is probed by D the first time. Since sending the address and content of a cells takes $2w$ bits, as long as $|P_A \cap P_B| \leq \epsilon_v k$, the last step will not be triggered, and the players follow the first four steps. Let us now focus on Step 4, query answering. Although Bob pretends that he has the right memory state, which might not always hold, indeed for all queries during which D does not probe any cell in P_A that is not in Alice's messages, Bob *will perform a correct simulation*. That is, as long as D does not probe any "unknown" cell in $P_A \cap P_B$, Bob will simulate D correctly. In the other words, each cell in $P_A \cap P_B$ can only lead to one incorrect query simulation among all k queries. When $|P_A \cap P_B| \leq \epsilon_v k$, on all but $\epsilon_v k$ queries, Bob's output agrees with the data structure. Thus, at least $(1/2 + \delta_v - \epsilon_v)$-fraction of the queries will be answered correctly, and this proves the lemma. □

To rule out the possibility of an efficient communication protocol for our problem, and prove Lemma 3.3, the main idea is to show that Bob has only learned very little information about the updates before each query Y_i. Alice's message can only depend on X and the previous queries, which are independent of Y_i. Thus, the probability that Bob answers each query correctly must be close to $1/2$. Finally, we obtain the desired probability bound from an application of the Azuma-Hoeffding inequality.

Proof of Lemma 3.3. Let R be the public random string, and M_i be Alice's message in Stage i. Let C_i be the indicator variable for correctly computing the i-th function $f(X, i, Y_i)$. We first show that until Stage i, Bob has learned very little about X even conditioned on $C_1, \ldots, C_{i-1}$, and thus could answer Y_i correctly with probability barely greater than $1/2$. Formally, we will prove by induction on i that

$$\Pr[C_i = 1 \mid C_1, \ldots, C_{i-1}] \leq \frac{1}{2} + O(\sqrt{\epsilon} + \sqrt{\epsilon_v}).$$

Fix a sequence $c_1, \ldots, c_{i-1} \in \{0, 1\}$. For simplicity of notation, denote the event $C_1 = c_1, \ldots, C_{i-1} = c_{i-1}$ by W_c. By induction hypothesis, we have $\Pr[W_c] \geq 2^{-O(i)} \geq 2^{-O(k)}$. Now conditioned on W_c, we upper bound the probability that P correctly answers the i-th query:

$$\begin{aligned}
&\Pr[P \text{ correctly computes } f(X, i, Y_i) \mid W_c] \\
&= \frac{1}{ns} \sum_{q=(l,b) \in [n] \times [s]} \Pr[P \text{ correctly computes } f(X, i, q) \mid W_c] \qquad (1)
\end{aligned}$$

Equality (1) is due to the fact that Y_i is uniform and independent of the previous inputs.

$$\leq \frac{1}{2} + \frac{1}{ns} \sum_{q=(l,b)\in[n]\times[s]} \mathop{\mathbb{E}}_{R,Y_1,\ldots,Y_{i-1},M_1,\ldots,M_i|W_c} \left| \Pr[f(X,i,q) = 1 \mid R, Y_1, \ldots, Y_{i-1}, M_1, \ldots, M_i, W_c] - \frac{1}{2} \right| \tag{2}$$

Inequality (2) holds because since Bob answers the query q based only on $R, Y_1, \ldots, Y_{i-1}, M_1, \ldots, M_i$, his advantage over $\frac{1}{2}$ of answering correctly is at most the bias of the conditional probability of $f(X, i, q)$.

$$\leq \frac{1}{2} + \frac{1}{ns} \sum_{q=(l,b)\in[n]\times[s]} \Theta\left(\sqrt{1 - H(f(X,i,q) \mid R, Y_1, \ldots, Y_{i-1}, M_1, \ldots, M_i, W_c)}\right) \tag{3}$$

Inequality (3) is due to Jensen's inequality and the fact that for a binary random variable Z such that $\Pr[Z = 1] = \frac{1}{2} \pm \epsilon$, its entropy is $H(Z) = 1 - \Theta(\epsilon^2)$.

$$\leq \frac{1}{2} + \Theta\left(\sqrt{\frac{1}{ns} \sum_{q=(l,b)\in[n]\times[s]} (1 - H(f(X,i,q) \mid R, Y_1, \ldots, Y_{i-1}, M_1, \ldots, M_i, W_c))}\right). \tag{4}$$

Finally, Inequality (4) is from another application of Jensen's inequality.

Furthermore, we have

$$\begin{aligned}
&\frac{1}{ns} \sum_{q=(l,b)\in[n]\times[s]} H(f(X,i,q) \mid R, Y_1, \ldots, Y_{i-1}, M_1, \ldots, M_i, W_c) \\
&\geq \frac{1}{ns} \sum_{l\in[n]} H(a_{\leq l}(X,i) \mid R, Y_1, \ldots, Y_{i-1}, M_1, \ldots, M_i, W_c) &(5)\\
&\geq \frac{1}{ns} \sum_{o\in[n/k]} H(a_{\leq o}(X,i), a_{\leq o+n/k}(X,i), \ldots, a_{\leq o+(k-1)n/k}(X,i) \mid R, Y_1, \ldots, Y_{i-1}, M_1, \ldots, M_i, W_c) &(6)
\end{aligned}$$

$a_{\leq t}(X,i)$ is the product of first t elements of a right before i-th query of I_B if the updates in I_A is X, i.e., the group element that $q = (t, *)$ queries. Inequality (5) and (6) is by the subadditivity of entropy and definition of the query function.

$$\geq \frac{1}{ns} \sum_{o\in[n/k]} (H(X \mid R, Y_1, \ldots, Y_{i-1}, M_1, \ldots, M_i, W_c) - s) \tag{7}$$

Inequality (7) is by our construction of the update sequence. The updates in I_A are evenly spaced. Thus, evenly spaced query can recover X (possibly except one element, which has entropy at most s).

$$\begin{aligned}
&= \frac{1}{ks} \cdot H(X \mid R, Y_1, \ldots, Y_{i-1}, M_1, \ldots, M_i, W_c) - \frac{1}{k} &(8)\\
&\geq \frac{1}{ks} \cdot (H(X \mid R, Y_1, \ldots, Y_{i-1}, W_c) - H(M_1, \ldots, M_i \mid R, Y_1, \ldots, Y_{i-1}, W_c)) - \frac{1}{k} &(9)
\end{aligned}$$

Inequality (9) is by the chain-rule for conditional entropy.

$$\geq \frac{1}{ks} \cdot H(X \mid R, Y_1, \ldots, Y_{i-1}, W_c) - O(\epsilon_v) - \frac{1}{k} \tag{10}$$

Inequality (10) is due to the fact that Alice sends no more than $O(\epsilon_v k \log n)$ bits and $s = \Theta(\log n)$.

$$\geq \frac{1}{ks} \cdot \left(k \log |G| - \log \frac{1}{\Pr[W_c]} \right) - O(\epsilon_v) - \frac{1}{k} \tag{11}$$

Inequality (11) is by the fact that X is uniform and independent of $R, Y_1, \ldots, Y_{i-1}$. For uniform X, we have $H(X \mid W) \geq H(X) - \log \frac{1}{\Pr[W]}$ for any event W.

$$\geq \frac{\log |G|}{s} - O\left(\frac{1}{s}\right) - O(\epsilon_v) - \frac{1}{k} \geq \frac{\log |G|}{s} - O(\epsilon_v + \epsilon), \tag{12}$$

Inequality (12) is by the induction hypothesis that $\Pr[W_c] \geq 2^{-O(k)}$ and $\epsilon \geq \Omega(1/\log n) \gg 1/k$.

Combining the above inequalities, we have

$$\begin{aligned} \Pr[P \text{ correctly computes } f(X, i, Y_i) \mid W_c] &\leq \frac{1}{2} + O\left(\sqrt{1 - \frac{\log |G|}{s} + O(\epsilon_v + \epsilon)} \right) \\ &\leq \frac{1}{2} + O\left(\sqrt{\epsilon} + \sqrt{\epsilon_v} \right). \end{aligned}$$

We have shown that conditioned on whether P successfully computes first $i-1$ function values, the probability that it succeeds on the next is always upper bounded by $\frac{1}{2} + O\left(\sqrt{\epsilon} + \sqrt{\epsilon_v}\right)$. Hence the random variables for the cumulative number of correct answers minus out cumulative upper bounds form a super-martingale, and we can apply the Azuma-Heoffding inequality [Hoe63]. The probability that $(1/2 + \delta_v - \epsilon_v)$-fraction of the function values are computed correctly is at most

$$\exp(-(\delta_v - O(\sqrt{\epsilon} + \sqrt{\epsilon_v}))^2 \cdot k).$$

This proves the lemma. □

3.2 Groups versus Monoids

One key property of groups needed for our proof is the invertibility. Consider generalizing to the *Monoid Range Problem*, which considers general monoids instead of groups. Monoids are sets closed under an associative operation and have an identity element (notice they do not have the invertibility property). We show that our lower bound does not hold for the Monoid Range Problem:

Theorem 1.4 (restated). *There exists a family of monoids $(G_n)_n$ such that the Monoid Range Problem can be solved in $O\left(\frac{\log n}{\log \log n}\right)$ time per operation.*

Proof. Consider the following family of monoids. We use $\times$ to denote the operator and 0 to denote the identity element, and $\star$ to denote a special element. The family has the following property: for any elements $x, y \in G_n$ we have that $x \times y = \star$ unless x or y is 0 (in which case their product equals the other, due to the identity property).

See Table 1 for small examples of these monoids. One way to think about these monoids is that the elements are zero, singletons, or products of more than one singleton ($\star$).

Thus, the product of a sequence of elements in G_n is $\star$ if there are more than one non-zero element; the product is 0 if all elements are zeros; the product is the only if non-zero element if there is exactly one. To efficiently maintain the range product of a G_n sequence, we use a *segment tree* of branching factor $B = \Theta(\log n)$.

$\times$	0	$\star$
0	0	$\star$
$\star$	$\star$	$\star$

$\times$	0	1	$\star$
0	0	1	$\star$
1	1	$\star$	$\star$
$\star$	$\star$	$\star$	$\star$

$\times$	0	1	2	$\star$
0	0	1	2	$\star$
1	1	$\star$	$\star$	$\star$
2	2	$\star$	$\star$	$\star$
$\star$	$\star$	$\star$	$\star$	$\star$

Table 1: Multiplication tables for G_2, G_3, and G_4.

The data structure Assume without loss of generality, that n is a power of B. Each node of the tree at depth i is associated with a (contiguous) subsequence of length n/B^i. Dividing the associated subsequence of a node E into B subsequences evenly, the j-th child of E is associated with the j-th subsequence. In particular, the root is associated with the entire sequence, the j-th child of the root is associated with $((j-1)\cdot n/B+1)$-th element to $(j\cdot n/B)$-th element, and each leaf is associated with a singleton. In each node E of the segment tree, the data structure maintains

(1) for each child of E, the minimum of two and the number of non-zero elements in their associated subsequences, i.e., if there is none or one or more than one non-zero element;

(2) if there is exactly one non-zero element in the associated subsequence of E, what this element is.

Note that Part (1) costs $O(1)$ bits for each child, thus $O(B) = O(\log n)$ bits in total, and Part (2) costs $O(\log n)$ bits to indicate the element. Thus, both parts can be stored in $O(1)$ words for each node.

Updates Upon receipt of an update $a_i := x$, the data structure iteratively updates the information in the tree bottom-up. It is not hard to verify that this update could only affect the nodes associated with some subsequence consisting of i. It first finds the leaf associated with $\{i\}$, and updates the two parts according to the value of x. Once all descendants of a node E are up-to-date, Part (1) of E can be updated by checking Part (1) of the only child of E affected by the update. From the updated Part (1) of E, one can figure out if there is exactly one non-zero element in the associated subsequence and which subtree it is in if there is. By checking Part (2) of the relevant subtree, the data structure will be able to update Part (2) of E. Updating each node takes $O(1)$ time, only $O(\log_B n)$ nodes are affected by the update. Thus, the total update time is $O(\log n/\log\log n)$.

Queries Recall that each child E_i of the root node is associated with a subsequence of length n/B. To answer the query $a_i\times\cdots\times a_j$, the data structure first breaks $[i,j]$ into subsequences $S_1, S_2, \ldots, S_m$, such that $S_2, \ldots, S_{m-1}$ are associated to $E_a, \ldots, E_{a+m-2}$ for some a, S_1 and S_m are subsequences of the associated subsequences of E_{a-1} and E_{a+m-1} respectively. By accessing Part (1) of the root node, the data structure learns whether there is none, exactly one or more than one non-zero elements in $S_2, \ldots, S_{m-1}$. Then it recurses on S_1 in the subtree rooted at E_{a-1} and S_m in the subtree rooted at E_{a+m-1}. By combining the answer from three parts, it will be able to output the answer to the query. It is not hard to verify that at each depth, at most two nodes of the tree may be recursed on. The query algorithm spends $O(1)$ time in each node. Thus, the total query time is $O(\log n/\log\log n)$.

Therefore, we conclude that the Monoid Range Problem with this particular family of monoids can be solved in $O(\log n/\log\log n)$ time per operation. □

4 The Matrix Range Problem

In this section, we show that for a particular group G, even maintaining one particular bit (say the last bit) of the whole product $\prod_{i=1}^{n} a_i$ is hard. The group G we focus on is the general linear group of invertible

matrices over the field $\mathbb{F}_p$ for constant p, namely $G = GL(\sqrt{\log n}, \mathbb{F}_p)$.

The binary encoding of matrices we would like to focus on is the encoding of a matrix as the concatenation of its entries. Hence, queries will return a bit about an entry of the matrix product. We call the Group Range Problem with $G = GL(\sqrt{\log n}, \mathbb{F}_p)$ and this encoding the *Matrix Range Problem*. However, since not all $\sqrt{\log n} \times \sqrt{\log n}$ matrices over $\mathbb{F}_p$ are invertible, this is not the most concise encoding of $GL(\sqrt{\log n}, \mathbb{F}_p)$. We remark that our desired encoding is nonetheless concise enough for Theorem 1.1 to hold:

Lemma 4.1. *Theorem 1.1 holds for the Matrix Range Problem.*

Proof. Consider the group of $\sqrt{\log n} \times \sqrt{\log n}$ invertible matrices over the field $\mathbb{F}_p$ where p is constant. Recall that this group has $|GL(\sqrt{\log n}, p)| = \prod_{i=0}^{\sqrt{\log n}-1}(p^n - p^i)$ elements (see e.g. [DF04, page 413]). We would like to represent this group in usual matrix format, i.e. as the concatenation of the representations of their entries. This representation uses $\log n \log p$ bits. On the other hand, notice that $\log |GL(n,p)| \geq (\log n - \sqrt{\log n}) \log p$, so Theorem 1.1 implies that our lower bound holds for this setting. □

The *Matrix Product Problem* is the same as the Matrix Range Problem, except that instead of being able to query for (a bit of) any entry of the product of the matrices in any subinterval, we are only allowed to query for (a bit of) the bottom-right entry of the product of the entire range of matrices. Despite this substantial restriction on the types of queries allowed, we find that the *Matrix Range Problem* can be reduced to the *Matrix Product Problem* such that our lower bounds from the previous section still apply to the *Matrix Product Problem*.

Lemma 4.2. *If the Matrix Product Problem for n matrices of dimension $d \times d$ can be solved in amortized $T(n,d)$ time per operation, then the Matrix Range Problem n matrices of dimension $d \times d$ can be solved in amortized $O(T(n, d+1))$ time per operation.*

Proof. The inspiration for the reduction is the following fact: Let $\mathbf{e}_i$ denote the i^{th} standard basis vector, i.e. the vector of length n whose entries are all 0 except for its i^{th} entry which is 1. For any $d \times d$ matrices A, B, and C, consider the following product of three $(d+1) \times (d+1)$ matrices:

$$D = \left[\begin{array}{c|c} A & \mathbf{0} \\ \hline \mathbf{e}_{j'}^T & 1 \end{array}\right] \left[\begin{array}{c|c} B & \mathbf{0} \\ \hline \mathbf{0}^T & 1 \end{array}\right] \left[\begin{array}{c|c} C & \mathbf{e}_{i'} \\ \hline \mathbf{0}^T & 1 \end{array}\right]$$

In the resulting matrix D, the bottom right entry $D_{(d+1)(d+1)}$ is equal to $(B_{i'j'} + 1)$.

The reduction is hence as follows. For any sequence $M_1, M_2, \ldots, M_n$ of $d \times d$ matrices, and any two indices $1 < i < j < n$, consider the following product of $(d+1) \times (d+1)$ matrices:

$$D = \left[\begin{array}{c|c} M_1 & \mathbf{0} \\ \hline \mathbf{0}^T & 1 \end{array}\right] \left[\begin{array}{c|c} M_2 & \mathbf{0} \\ \hline \mathbf{0}^T & 1 \end{array}\right] \cdots \left[\begin{array}{c|c} M_{i-1} & \mathbf{0} \\ \hline \mathbf{e}_{j'}^T & 1 \end{array}\right] \left[\begin{array}{c|c} M_i & \mathbf{0} \\ \hline \mathbf{0}^T & 1 \end{array}\right] \cdots \left[\begin{array}{c|c} M_j & \mathbf{0} \\ \hline \mathbf{0}^T & 1 \end{array}\right] \left[\begin{array}{c|c} M_{j+1} & \mathbf{e}_{i'} \\ \hline \mathbf{0}^T & 1 \end{array}\right] \cdots \left[\begin{array}{c|c} M_n & \mathbf{0} \\ \hline \mathbf{0}^T & 1 \end{array}\right]$$

Similar to before, the bottom right entry $D_{(d+1)(d+1)}$ will be equal to the $(i', j')^{th}$ entry of the product $M_i \cdots M_j$ plus one. To deal with $i = 1$, then no matrix has $\mathbf{e}_{j'}^T$ as its bottom row. The first d entries of the right column of D will be the i'^{th} column of $M_1 \cdots M_j$. There is a similar case for $j = n$.

Updates to the original sequence of $d \times d$ matrices can be translated directly into updates to the new sequence of $(d+1) \times (d+1)$ matrices. Queries to the original sequence result in at most four updates and a query on the new sequence. This completes the proof. □

Corollary 1.3 (restated). *Theorem 1.1 holds for the Matrix Product Problem.*

4.1 Upper Triangular Matrices

We further restrict our focus to the group G of invertible upper triangular matrices. In some applications, only upper triangular matrices are sufficient instead of the full general linear group of all invertible matrices, and the proof of Lemma 4.2 does not immediately imply that the *Upper Triangular Matrix Product Problem* has a Theorem 1.1 style of lower bound, as our gadget would make one matrix no longer upper triangular. Nonetheless, we are able to prove the lower bound via a modification of Lemma 4.2.

Lemma 4.3. *If the Upper Triangular Matrix Product Problem for n matrices of dimension $d \times d$ can be solved in amortized $T(n, d)$ time per operation, then the Matrix Range Problem n matrices of dimension $d \times d$ can be solved in amortized $O(T(2n + 1, 2d))$ time per operation.*

Proof. The reduction uses the following identity: Let $N_{(i,j)}$ denote the $d \times d$ matrix which has all entries 0 except its (i, j) entry is 1. For any $d \times d$ upper triangular matrices A, B, and C, we have the following identity of $2d \times 2d$ upper triangular matrices:

$$\left[\begin{array}{c|c} A & 0 \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} N_{(1,j)} & 0 \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} B & 0 \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} I & N_{(i,1)} \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} C & 0 \\ \hline 0 & I \end{array}\right] = \left[\begin{array}{c|c} AB_jC & B_{(i,j)} \\ \hline 0 & I \end{array}\right],$$

where B_j is the all zeroes matrix except that its first row is the jth row of B, and $B_{(i,j)}$ is the all zeroes matrix except that its top right entry is the (i, j) entry of B.

Similar to before, to maintain the sequence $M_1, \ldots, M_n$ of $d \times d$ matrices, we will maintain the following sequence of $2d \times 2d$ matrices:

$$\left[\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} M_1 & 0 \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} M_2 & 0 \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right] \cdots \left[\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} M_n & 0 \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right].$$

To query the (i, j) entry of the product $M_aM_{a+1} \cdots M_b$, we change the $(2a - 1)$th matrix to $\left[\begin{array}{c|c} N_{(1,j)} & 0 \\ \hline 0 & I \end{array}\right]$, and change the $(2b + 1)$th matrix to $\left[\begin{array}{c|c} I & N_{(i,1)} \\ \hline 0 & I \end{array}\right]$, and then our desired value is the top right entry of the product of all the matrices.

□

Corollary 4.4. *Theorem 1.1 holds for the Upper Triangular Matrix Product Problem.*

References

[Ajt88] Miklós Ajtai. A lower bound for finding predecessors in yao's call probe model. *Combinatorica*, 8(3):235–247, 1988.

[BF02] Paul Beame and Faith E. Fich. Optimal bounds for the predecessor problem and related problems. *J. Comput. Syst. Sci.*, 65(1):38–72, 2002.

[BH11] George F Burkhard and Eric T Hoke. Transfer matrix optical modeling. 2011.

[DF04] David Steven Dummit and Richard M Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[HW07] Johan Håstad and Avi Wigderson. The randomized communication complexity of set disjointness. *Theory of Computing*, 3(1):211–219, 2007.

[LD69] YK Lin and BK Donaldson. A brief survey of transfer matrix techniques with special reference to the analysis of aircraft panels. *Journal of Sound and Vibration*, 10(1):103–143, 1969.

[LW17] Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In *SODA*, pages 2182–2189, 2017.

[MNSW95] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. In *STOC*, pages 103–111, 1995.

[Păt07] Mihai Pătraşcu. Lower bounds for 2-dimensional range counting. In *STOC*, pages 40–46, 2007.

[PD04] Mihai Pătraşcu and Erik D Demaine. Tight bounds for the partial-sums problem. In *SODA*, pages 20–29, 2004.

[PD06] Mihai Pătraşcu and Erik D Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

[PRI99] Leif AA Pettersson, Lucimara S Roman, and Olle Inganäs. Modeling photocurrent action spectra of photovoltaic devices based on organic thin films. *Journal of Applied Physics*, 86(1):487–496, 1999.

[PT11] Mihai Pătraşcu and Mikkel Thorup. Don't rush into a union: take time to find your roots. In *STOC*, pages 559–568, 2011.

[PYF03] Peter Peumans, Aharon Yakimov, and Stephen R Forrest. Small molecular weight organic thin-film photodetectors and solar cells. *Journal of Applied Physics*, 93(7):3693–3723, 2003.

[WY16] Omri Weinstein and Huacheng Yu. Amortized dynamic cell-probe lower bounds from four-party communication. In *FOCS*, pages 305–314, 2016.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *FOCS*, pages 222–227, 1977.

[Yao81] Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM (JACM)*, 28(3):615–628, 1981.

[Yu16] Huacheng Yu. Cell-probe lower bounds for dynamic problems via a new communication model. In *STOC*, pages 362–374, 2016.

A Applications of the Group Range Problem

Physics

One example in the areas of optics and computer graphics is the propagation of electromagnetic waves through different media. The *transfer-matrix method* from optics describes how to analyze the propagation of such waves by computing a product of *characteristic matrices*, one corresponding to each medium. In optical modeling experiments, physicists sometimes need to quickly determine how making changes to one characteristic matrix alters the overall product, a computational task described by our problem. [PRI99, PYF03, BH11] Some forms of ray tracing in computer graphics also use this transfer matrix analysis.

The transfer matrix method is also used in some mechanical engineering problems, like in the design of aircraft panels. Due to the details of these mechanics problems, the matrices involved are typically upper-triangular matrices. [LD69]

In many of these applications, one is only interested in being able to query the product of the entire sequence of matrices, rather than querying arbitrary subintervals of matrices. We show that, when G is the group of invertible matrices, or the group of invertible upper triangular matrices, our lower bound still holds even if only an entry of the product of the entire sequence of matrices can be queried.

Dynamic Permanent for Banded Matrices

Consider the following data structure problem. We want to keep track of the permanent of an $n \times n$ matrix M over some finite field $\mathbb{F}_p$. To keep the problem tractable (because Permanent is NP-complete), we restrict attention to the case where M is a band matrix, i.e. $M_{i,j}$ is nonzero only when $|i-j| \leq k$ for some constant k. We want to support the following two operations:

- Update(i, j, Δ), which updates $M_{i,j} \leftarrow M_{i,j} + \Delta$ but only for $|i-j| \leq 1$.
- Query(), which returns the permanent of M.

It turns out that this problem is reducible to the Matrix Range Problem. Consider the $k = 1$ case, and treat the permanent as the sum of weights of perfect matchings of a bipartite graph $G = ([n], [n], E)$, we define P_i to be the sum of weights of perfect matchings of the bipartite graph $G_i = ([i], [i], E \cap ([i] \times [i]))$. Because of the banded property of the matrix, there are only two vertices that vertex i on the left hand side of the graph can be matched to: vertex $i-1$ or vertex i on the right hand side. Furthermore, if left i is matched to right $i-1$ then left $i-1$ must be matched to right i. Hence:

$$P_i = P_{i-1}M_{i,i} + P_{i-2}M_{i-1,i}M_{i,i-1}$$

$$\begin{bmatrix} P_i \\ P_{i-1} \end{bmatrix} = \begin{bmatrix} M_{i,i} & M_{i-1,i}M_{i,i-1} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_{i-1} \\ P_{i-2} \end{bmatrix}$$

$$\begin{bmatrix} P_n \\ P_{n-1} \end{bmatrix} = \begin{bmatrix} M_{n,n} & M_{n-1,n}M_{n,n-1} \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} M_{2,2} & M_{1,2}M_{2,1} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} M_{1,1} \\ 1 \end{bmatrix}$$

Therefore it suffices to keep a Matrix Range which stores the $n-1$ matrices $\begin{bmatrix} M_{i,i} & M_{i-1,i}M_{i,i-1} \\ 1 & 0 \end{bmatrix}$. Each update to the Dynamic Permanent data structure results in exactly one update to the Matrix Range data structure, and a query to the Dynamic Permanent data structure can be answered by querying for the top row (two entries) of the product of the entire range.

Modulo the fact that these matrices may not be invertible (if $M_{i-1,i}M_{i,i-1} = 0$), our results show that this approach to the problem should cost $\Omega(\log n)$ time per operation. In particular, our lower bound for the Matrix Product Problem showed that this problem is still hard even when queries only request the entire range of matrices and not arbitrary subintervals.