Randomization Does Not Help Searching Predecessors

Mihai Pătraşcu
mip@mit.edu

Mikkel Thorup
mthorup@research.att.com

Abstract

At STOC'06, we presented a new technique for proving cell-probe lower bounds for static data structures with deterministic queries. This was the first technique which could prove a bound higher than communication complexity, and it gave the first separation between data structures with linear and polynomial space. The new technique was, however, heavily tuned for the deterministic worst-case, demonstrating long query times only for an exponentially small fraction of the input. In this paper, we extend the technique to give lower bounds for randomized query algorithms with constant error probability.

Our main application is the problem of searching predecessors in a static set of n integers, each contained in a ℓ-bit word. Our trade-off lower bounds are tight for any combination of parameters. For small space, i.e. $n^{1+o(1)}$, proving such lower bounds was inherently impossible through known techniques. An interesting new consequence is that for near linear space, the classic van Emde Boas search time of $O(\lg \ell)$ cannot be improved, even if we allow randomization. This is a separation from polynomial space, since Beame and Fich [STOC'02] give a predecessor search time of $O(\lg \ell / \lg \lg \ell)$ using quadratic space.

We also show a tight $\Omega(\lg \lg n)$ lower bound for 2-dimensional range queries, via a new reduction. This holds even in rank space, where no superconstant lower bound was known, neither randomized nor worst-case. We also slightly improve the best lower bound for the approximate nearest neighbor problem, when small space is available.

1 Introduction

The main result of this paper is a randomized lower bound for static predecessor search, matching the best deterministic upper bounds, thus showing that the current query time cannot be improved using randomization. This stands in contrast to the situation for problems such as high dimensional approximate nearest neighbors where randomization provides a provable advantage.

The predecessor search problem is to represent an ordered set Y such that for any query x we can find efficiently $\texttt{PREDECESSOR}(x) = \max\ \{y \in Y \mid y \leq x\}$. This would be the predecessor of x in a sorted list over $\{x\} \cup Y$. Predecessor search is one of the most fundamental and well-studied problems in data structures. For a comprehensive list of references, we refer to [3]; here, we only describe briefly the best known bounds.

1.1 The Upper-Bound Story.

We focus on the static case, where Y is given in advance for preprocessing. For example, we can sort Y, and later find the predecessor of x by binary search using $O(\lg n)$ comparisons, where $n = |Y|$.

On computers, we are particularly interested in integer keys. Thereby we also handle, say, floating point numbers whose ordering is preserved if they are cast as integers. We can then use all the instructions on integers available in a standard programming language such as C, and we are no longer limited by the $\Omega(\lg n)$ comparison based lower bound for searching. A strong motivation for considering integer keys is that integer predecessor search is asymptotically equivalent to the IP look-up problem for forwarding packets on the Internet [6]. This problem is extremely relevant from a practical perspective. The fastest deployed software solutions use non-comparison-based RAM tricks (see e.g. [5]).

More formally, we represent Y on a word RAM with a given word length b. We assume each integers in Y has ℓ bits, and that $\lg n \leq \ell \leq b$. On the RAM, the most natural assumption is $\ell = b$. The case $b > \ell$ models the external memory model with $B = \lfloor \frac{b}{\ell} \rfloor$ keys per page. In this case, the well-known (comparison-based) B-trees achieve a search time of $O(\log_B n)$. Unless otherwise stated, we assume below $b = \ell$.

Using the classic data structure of van Emde Boas [14] from 1975, we can represent our integers so that predecessors can be searched in $O(\lg \ell)$ time. The space is linear if we use hashing [15]. In the 1990, Fredman and Willard [7] introduced fusion trees, which requires linear space and can answer queries in $O(\log_\ell n)$ time. Combining with van Emde Boas' data structure, they got a search time of $O(\min\{\frac{\lg n}{\lg \ell}, \lg \ell\})$, which is always $O(\sqrt{\lg n})$.

In 1999, Beame and Fich [3] found an improvement to van Emde Boas' data structure bringing the search time down to $O(\frac{\lg \ell}{\lg \lg \ell})$. Combined with fusion trees, this gave them a bound of $O(\min\{\frac{\lg n}{\lg \ell}, \frac{\lg \ell}{\lg \lg \ell}\})$, which is always $O(\sqrt{\frac{\lg n}{\lg \lg n}})$. However, the new data structure of Beame and Fich uses quadratic space, and they asked if the space could be improved to linear or near-linear.

Finally, at STOC'06 [12], we presented an optimal deterministic trade-off. Define $\lg x = \lceil \log_2(x+2) \rceil$, so that $\lg x \geq 1$ even if $x \in [0,1]$. Assuming S bits of space are available, and defining $a = \lg \frac{S}{n}$, we showed that the optimal search time is, up to constant factors:

$$\min \begin{cases} \log_b n \\ \lg \frac{\ell - \lg n}{a} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\frac{a}{\lg n} \cdot \lg \frac{\ell}{a}\right)} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\lg \frac{\ell}{a} \Big/ \lg \frac{\lg n}{a}\right)} \end{cases} \tag{1.1}$$

We refer to [12] for a discussion of the significance of each branch of this trade-off. The most important consequence is that for a polynomial universe with $b = \ell = \Theta(\lg n)$ and near-linear space $S = \widetilde{O}(n)$ [1] , the classic van Emde Boas bound of $O(\lg \ell) = O(\lg \lg n)$ is optimal. Such polynomial universes often appear inside combinatorial algorithms.

These upper bounds are achieved by a deterministic query algorithm on a RAM. The data structure can be constructed in expected time $O(S)$ by a randomized algorithm, starting from a sorted list of integers.

1.2 The Lower-Bound Story. Ajtai [1] was the first to prove a superconstant lower bound for our problem. His results, with a correction by Miltersen [9], can be interpreted as saying that there exists n as a function of ℓ such that the time complexity for polynomial space is $\Omega(\sqrt{\lg \ell})$, and likewise there exists ℓ as a function of n making the time complexity $\Omega(\sqrt[3]{\lg n})$.

Miltersen [9] revisited Ajtai's work, showing that the lower bound holds in the communication game model, and for a simpler colored predecessor problem. In this problem, the elements of Y have an associated color (say, red or blue), and the query asks only for the color of the predecessor in Y. This distinction is important, as one can reduce other problems to this simpler problem, such as existential range queries in two dimensions [10] or prefix problems in a certain class of monoids [9]. Like previous lower bound proofs, ours also holds for the colored problem, making the lower bounds applicable to these problems.

Miltersen, Nisan, Safra and Wigderson [10] once again revisited Ajtai's proof, extending it to randomized algorithms. More importantly, they captured the essence of the proof in an independent *round elimination lemma*, which forms a general tool for proving communication lower bounds. Our cell-probe elimination lemma is inspired, at a high level, by this result.

[1]We define $\widetilde{O}(n) = n \lg^{O(1)} n$.

Beame and Fich [3] improved the lower bounds to $\Omega(\frac{\lg \ell}{\lg \lg \ell})$ and $\Omega(\sqrt{\frac{\lg n}{\lg \lg n}})$ respectively. Sen and Venkatesh [13] later gave an improved round elimination lemma, which can reprove the lower bounds of Beame and Fich, but also for randomized algorithms. The time-space trade-offs obtained by these proofs are $\Omega(\frac{\lg n}{\lg b}, \frac{\lg \ell}{\lg \lg S})$, where S is the space bound, and possibly $b > \ell$.

At STOC'06 [12], we presented lower bounds for a deterministic querier matching the upper bounds in (1.1). When $S \geq n^{1+\varepsilon}$ for some constant $\varepsilon > 0$, the lower bounds could be shown via the stronger communication game model, and allowed randomization with two-sided error. The most interesting part, however, was the improved lower bound for space $n^{1+o(1)}$. These were the first bounds surpassing communication complexity, and they provided the first separation between linear and polynomial space for *any* data structure problem. Unfortunately, these bounds only held for the deterministic case.

2 Our Contributions

The main result of this paper is that the deterministic predecessor bound from (1.1) cannot be improved even if we allow randomization. As mentioned above, the unresolved case was with small space $n^{1+o(1)}$, which is the most interesting case in practice.

Our previous deterministic lower bound [12] for small space was heavily tuned for the deterministic worst-case, demonstrating long query times only for a fraction of the input which is exponentially small in the query time. In this paper, we generalize the technique to give lower bounds for randomized query algorithms with constant error probability.

2.1 Randomization versus Determinism. In this paper, "randomized algorithm" refers to an algorithm with a fixed running time, which may make a two-sided error with probability bounded away from 1/2. This model is stronger that one-sided or zero-error (Las Vegas) randomization, so lower bounds also apply in these settings.

Traditionally, proving a randomized lower bound is considered a challenging task, even if the deterministic bound is known and believed to hold even with randomization. Our own predecessor problem offers several examples in which it took much technical progress until a randomized lower bound was given. Ajtai [1] published the first deterministic lower bound in 1988, but it was not until STOC'95 that Miltersen et al [10] gave the same randomized bound. In STOC'99, Beame and Fich [3] proved an optimal deterministic bound for polynomial space, but it took until ICALP'01 and CCC'03 [13] for this to be extended to the randomized case by Sen and Venkatesh.

It is also interesting to note that for a related problem, approximate nearest neighbor searching, known deterministic lower bounds are much higher than randomized upper bounds [8], showing that randomization *can* help crucially for natural data-structure problems. We also consider this problem below, and describe an improvement to the randomized lower bound.

In the course of proving the deterministic lower bounds for the predecessor problem, we introduced a new concept which was crucial to the induction hypothesis: we allowed the algorithm to *reject* queries, under certain conditions. In fact, the previous proof rejects almost all queries; nonetheless the few accepted queries remaining carry enough information to contradict a fast query time. We note that which queries are accepted depends non-trivially on the data structure and query algorithm. This dependence was not a problem for the proof, as any accepted query had to be handled correctly by the deterministic query algorithm.

However, in the randomized error case, it could be that we only accept queries leading to errors. This is not known a priori, since the errors are silent. A new challenge here is hence to make sure that accepted queries to not have a particularly high error rate (i.e., conditioned on the query being accepted, this algorithm still is correct with good probability). Since the concept of rejects was a key to our previous progress, it seems essential to study the interplay between errors and rejects. We conjecture these ideas will prove significant beyond the scope of our current results.

2.2 Not the first randomized separation. As we are randomizing the first deterministic separation between linear and polynomial space for *any* data structure problem, it seems that this should be the first such randomized separation. However, very recently [11], we presented such a separation for so-called rich problems, like exact high dimensional nearest neighbors. The separation worked for any rich problem, both deterministically and randomized. However, the predecessor problem is known to be far from rich, so the results and proof techniques of [11] are irrelevant here.

2.3 Approximate Nearest Neighbor Search. In FOCS'04, Chakrabarti and Regev [4] showed the first lower bound for randomized $O(1)$-approximate nearest neighbor search. If points come from the d-dimensional Hamming cube, they showed that a data structure using $n^{O(1)}$ words of $d^{O(1)}$ bits requires $\Omega(\lg\lg d/\lg\lg\lg d)$ query time. This result is tight for large enough polynomial space, but it is not known how to attain such good query times with close to linear space (an important practical problem). Using our techniques, one can show an $\Omega(\lg\lg d)$ time lower bound given $\widetilde{O}(n)$ space, which, however, is unlikely to be optimal. Note that for deterministic algorithms, a much higher lower bound of $\Omega(d/\lg n)$ is known [8]. Thus, it is essential that we can prove a randomized bound.

2.4 Range Queries in Two Dimensions. Another problem closely related to predecessor search is static range searching in two dimensions. Given a set of n points at integer coordinates in the plane, the query asks whether an axis-parallel rectangle contains any point. Consider the colored predecessor problem, where elements of the set Y are red or blue, and the query should only return the color of the predecessor. Lower bounds for this problem (such as ours) also apply to range queries, as observed by [10]. The trick is to consider the interval stabbing problem, where intervals are define by a red point and the next blue point. This problem is itself easily reducible to 2D range searching (even to the special case of dominance queries, where one corner of the rectangle is always the origin).

An interesting special case is when coordinates are distinct integers in $[n]$, i.e. the problem is in rank space. This restriction occurs naturally in many important cases, such as recursion from higher-dimensional range structures, or geometric approaches to pattern matching. In FOCS'00, Alstrup et al.[2] gave a query time of $O(\lg\lg n)$, using space $O(n\lg^{\varepsilon} n)$. Clearly, predecessor lower bounds are irrelevant, since predecessors in $[n]$ are trivial to find with $O(n)$ space. In fact, no previous technique could prove a superconstant lower bound.

We can show a tight $\Omega(\lg\lg n)$ time bound for space $\widetilde{O}(n)$. Note that this requires a novel approach, since for dominance queries, as obtained by the old reduction, there is a constant-time upper bound (the RMQ data structures). In a nutshell, we consider a uniform $\sqrt[3]{n}\times\sqrt[3]{n}$ grid on top of our original space. In each cell, we construct a hard subproblem using the colored predecessor problem. This is possible since we get to place $\sqrt[3]{n}$ points in the space $\left[n^{2/3}\right]^2$. Finally, we can use the direct-sum properties of our lower bound, to argue that for this set of problems, the query time cannot be better than for one problem with $\sqrt[3]{n}$ points and $\widetilde{O}(\sqrt[3]{n})$ space. Further details are given in the full version.

2.5 An extended abstract. This paper needs to be understood as an extended abstract. Essentially, it contains only a description of the new ideas needed for the proof. We only briefly summarize the portions of the old proof which we use (which are themselves rather involved), and refer to [12] for details about those.

3 Randomized Cell-Probe Elimination

3.1 Statement of the Central Lemma. An abstract decision data structure problem is defined by a function $f : D \times Q \to \{0, 1\}$. An input from D (e.g. a set of n integers) is given at preprocessing time, and the data structure must store a representation of it in some bounded space. An input from Q (e.g. an integer to search for) is given at query time, and the function of the two inputs must be computed.

As mentioned before, we work in the cell-probe model, and let b be the number of bits in a cell. We assume the query's input consists of at most b bits, and that the space bound is at most 2^b. For the sake of an inductive argument, we extend the cell-probe model by allowing the data structure to publish some bits at preprocessing time. These are bits depending on the data structure's input, which the query algorithm can inspect at no charge.

The query algorithm may not compute f correctly in two cases. First, we allow the algorithm to *reject* some queries, in the following limited way. After inspecting the query and the published bits, the algorithm can decide to not answer the query (a reject). Otherwise, we say the query is accepted; the algorithm can make cell probes, and at the end it must produce an answer. Later rejects are disallowed. In contrast to silent error, it makes sense to talk about tiny (close to zero) probabilities of accepting, even for problems with boolean output.

If the query is accepted, the algorithm may still produce an incorrect output, which we call an *error*. We allow silent, two-sided errors. We define the error probability to be the probability the output is incorrect, conditioned on the fact that the query is accepted. Naturally, the problem is trivial if the error probability is allowed to be $1/2$, regardless of the accept probability.

We always consider a problem in conjunction with a distribution $\mathcal{D}$ over $D \times Q$, and we restrict the preprocessing and query algorithms to be deterministic. The reject and error probabilities are defined under $\mathcal{D}$. We can use the easy direction of Yao's minimax principle [16] to obtain bounds for randomized algorithms. Considering a classic cell-probe algorithm which is not allowed to reject queries, we can fix its random bits so that the error probability is maintained under $\mathcal{D}$. We can later manipulate the algorithm in our model (possibly introducing rejects).

For an arbitrary problem f and an integer $k \le 2^b$, we define a direct-sum problem $\bigoplus^k f : D^k \times ([k] \times Q) \to \{0, 1\}$ as follows. The data structure receives a vector of inputs $(d^1, \dots, d^k)$. The representation depends arbitrarily on all of these inputs. The query is the index of a subproblem $i \in [k]$, and an element $q \in Q$. The output of $\bigoplus^k f$ is $f(d^i, q)$. We also define a distribution $\bigoplus^k \mathcal{D}$ for $\bigoplus^k f$, given a distribution $\mathcal{D}$ for f. Each d^i is chosen independently at random from the marginal distribution on D induced by $\mathcal{D}$. The subproblem $i \in [k]$ is chosen uniformly, and q is chosen from the distribution on Q conditioned on d^i.

Given an arbitrary problem f and an integer $h \le b$, we can define another problem $f^{(h)}$ as follows. The query is a vector $(q_1, \dots, q_h)$. The data structure receives a regular input $d \in D$, an integer $r \in [h]$ and the prefix of the query $q_1, \dots, q_{r-1}$. The output of $f^{(h)}$ is $f(d, q_r)$. Note that we have shared information between the data structure and the querier (i.e. the prefix of the query), so $f^{(h)}$ is a partial function on the domain $D \times \bigcup_{i=0}^{h-1} Q^i \times Q$. Now we define an input distribution $\mathcal{D}^{(h)}$ for $f^{(h)}$, given an input distribution $\mathcal{D}$ for f. The value r is chosen uniformly at random. Each query coordinate q_i is chosen independently at random from the marginal distribution on Q induced by $\mathcal{D}$. Now d is chosen from the distribution on D, conditioned on q_r. We give the $f^{(h)}$ operator precedence over the direct sum operator, i.e. $\bigoplus^k f^{(h)}$ means $\bigoplus^k \left[f^{(h)}\right]$.

Now consider an algorithm $\mathcal{A}$ for some problem $\bigoplus^k f$. For some instance p of the published bits, a subproblem index $i \in [k]$, and a query $q \in Q$ the algorithm may decide to reject the query or begin by probing a certain cell. We define $\mathcal{A}^i(p; q)$ to be the cell probed, or $\dagger$ if the query is rejected. Thus, we use $\mathcal{A}$ as a function giving the first action taken by the algorithm. We also write $\mathcal{A}^i(p) = \{\mathcal{A}^i(p; q) \mid q \in Q\}$ and $\mathcal{A}^*(p) = \bigcup_{i \in [k]} \mathcal{A}^i(p)$. That is, $\mathcal{A}^*(p)$ is the set of all possible initial cell probes that the algorithm could make when the published bits are p.

Lemma 3.1. *There exists a universal constant C, such that for any problem f, distribution $\mathcal{D}$ and $\varepsilon_\uparrow > 0$, the following holds in the cell-probe model with $\frac{(\alpha\varepsilon_\uparrow)^6}{C} k$ published bits allowed. Assume there exists a solution $\mathcal{A}$ to $\bigoplus^k f^{(2)}$ with accept probability α and error probability ε over $\bigoplus^k \mathcal{D}^{(2)}$, which satisfies $(\forall) p : |\mathcal{A}^*(p)| \le k\sigma$, for some σ. Then there exists a solution $\mathcal{B}$ for $\bigoplus^k f$, making the same number of probes as $\mathcal{A}$, and satisfying one of the following two sets of properties:*

1. *accept probability $\frac{\alpha\varepsilon_\uparrow}{C}$ and error probability $\varepsilon + \varepsilon_\uparrow$ over $\bigoplus^k \mathcal{D}$, as well as $(\forall) p : |\mathcal{B}^*(p)| \le k\sqrt{\sigma} \cdot Cb$.*

2. *for some integer $j \ge 1$, accept probability $\alpha\varepsilon_\uparrow / C^j$ and error probability $\varepsilon - \frac{j}{C}\varepsilon_\uparrow$ over $\bigoplus^k \mathcal{D}$, as well as $(\forall) p : |\mathcal{B}^*(p)| \le k\sigma$.*

Note that applying the lemma yields an improvement in one of two directions: it reduces either the error, or the space of initial cell probes. Once the initial cell probe is in a small enough set, one can simply include those cells among the published bits, and skip the first cell probe.

3.2 Applying the lemma to the predecessor problem. We now sketch the application of Lemma 3.1 to predecessor search. Since this is rather standard, we present a compact summary of what is done in [12], and highlight the interesting differences. The rest of the paper concentrates on actually proving Lemma 3.1, which needs the most novel ideas.

First, we sketch the proof of our deterministic bounds in [12]. Let $P(n, \ell)$ be the colored predecessor problem on n keys of length ℓ. Suppose we are dealing with a direct sum problem $\bigoplus^k P(n, h\ell)$. One first shows that a solution for $\bigoplus^k P(n, h\ell)$ is also a solution for $\bigoplus^k P(n, \ell)^{(h)}$. Given a solution to $\bigoplus^k P(n, \ell)^{(h)}$, one can apply a lemma similar to our Lemma 3.1, and eliminate the first cell probe. This changes the problem to $\bigoplus^k P(n, \ell)$, i.e. reduces the key length by a factor of h. It also introduces roughly $k\sqrt[h]{\sigma}$ published bits. To balance that, one amplifies the number of subproblems by roughly $t = \sqrt[h]{\sigma}$: it can be shown that a solution for $\bigoplus^k P(n, \ell)$ is also a solution for $\bigoplus^{kt} P(n/t, \ell - \lg t)$. After this, the argument is repeated. In the end, we are left with a nontrivial problem, but zero cell-probes. An information-theoretic argument shows a contradiction. This argument goes through even if the accept probability is rather small, so it is not a problem if the accept rate drops geometrically in every step.

Note that the hard distribution is constructed implicitly by this argument. Eliminating a cell probe when going from $\bigoplus^k P(n, \ell)^{(h)}$ to $\bigoplus^k P(n, \ell)$ can also be viewed as constructing a hard distribution $\bigoplus^k \mathcal{D}^{(h)}$ for $\bigoplus^k P(n, \ell)^{(h)}$, starting from whatever distribution $\bigoplus^k \mathcal{D}$ is hard for $\bigoplus^k P(n, \ell)$. Thus, to obtain a hard distribution for the original problem, start with a (trivial) distribution which is hard for zero cell probes, and follow the cell-probe eliminations in reverse.

Let us now describe how this overall argument is modified for a randomized lower bound. The essential difference is that we need to use our new Lemma 3.1 to eliminate a cell probe. In the scenario of $h = 2$, we simply apply the lemma once. If case 1 happens, $|\mathcal{B}^*(p)| = O(k\sqrt{\sigma}w)$. Then, the data structure can publish, in addition to p (its old published bits), the contents of all cells in $\mathcal{B}^*(p)$. The algorithm can simply skip the first cell probe, and retrieve the data from the published bits. Thus, if we only require $h = 2$ and case 1 of the lemma always comes out, the proof is unchanged.

The first issue that we have to face is that some branches of the tradeoff require $h > 2$. Up to constant factors, it suffices to take h a power of two. Then, we need $\log_2 h$ applications of the lemma leading to case 1. After this, the space of initial cell probes is reduced to less than $k\sqrt[h]{\sigma}(Cb)^2$, so we can simply publish all these cells. This is an interesting trick which allows us to prove the lemma just for $f^{(2)}$. In the deterministic case, it was not hard to prove a similar lemma directly for $f^{(h)}$, but the complications of the randomized bounds make this infeasible.

The second issue we address is the possibility of case 2. If this happens, we simply keep applying the lemma until we see case 1. Let us assume that the initial algorithm has an error of at most 1/3, and let T be the number of eliminations we want to perform. Then, we use the lemma with $\varepsilon_\uparrow = \frac{1}{9T}$. We apply the lemma until case 1 happens T times, so the maximum error we can get to is $\frac{1}{3} + \frac{1}{9} < \frac{1}{2}$. Then, the problem remains nontrivial at all times. Now note that case 2 can only happen $O(T)$ times, because each time it happens, the error is reduced by $\Omega(1/T)$. Thus, case 2 can be amortized away against the useful case 1, and the lower bound is only decreased by a constant factor.

The final worry is that the accept probability can decrease arbitrarily in case 2. However, if the probability decreases by C^j, the error also decreases by $\Omega(j/T)$. Then, by an amortization as above, the accept probability cannot decrease past $2^{-O(T)}$.

4 Proof of the Lemma 3.1

4.1 Preliminaries. We first introduce some convenient notation. Remember that the data structure's input for $\bigoplus^k f^{(2)}$ consists of a vector $(d^1, \dots, d^k) \in D^k$, a vector selecting the interesting segments $(r^1, \dots, r^k) \in [2]^k$ and the query prefixes $Q^i \in Q$, for each i with $r^i = 2$ (we can take Q^i to be the empty string when $r^i = 1$). Denote by $\mathbf{d}, \mathbf{r}$ and $\mathbf{Q}$ the random variables describing these three components of the input. Also let $\mathbf{p}(r, Q, d)$ be the function representing the bits published by the data structure for a specific input. The query consists of an index i selecting the interesting subproblem, and a vector (q_1, q_2) with a query to that subproblem. Denote by $\mathbf{i}$ and $\mathbf{q}$ these random variables. Note that in our probability space $\bigoplus^k \mathcal{D}^{(2)}$, we have $\mathbf{q}_1 = \mathbf{Q}^{\mathbf{i}}$ whenever $\mathbf{r}^{\mathbf{i}} = 2$.

For an instance p of the published bits, and a prefix q_1 for a query to subproblem i, we can make the following definitions:

$\Psi^i(p; q_1) = \{\mathcal{A}^i(p; q_1, q_2) \mid q_2 \in Q\} \setminus \{\dagger\}$. Note also that the marginal distribution of $q_2 \in Q$ induces a distribution on the support $\Psi^i(p;q_1)\cup\{\dagger\}$.

$T^i(p;q_1) =$ the set of $\sqrt{\sigma}$ cells in $\Psi^i(p;q_1)$ which have the highest probability of being probed (or all cells if $|\Psi^i(p;q_1)| < \sqrt{\sigma}$). Again, our probability space is given by a random choice of q_2 from the marginal distribution on Q.

$\tau^i(p;q_1) = \Pr_{\mathbf{q}_2}[\mathcal{A}^i(p;q_1,q_2)\in T^i(p;q_1) \mid \mathcal{A}^i(p;q_1,q_2)\neq\dagger]$. In other words, $\tau^i(p;q_1)$ is the probability density of $T^i(p;q_1)$ in $\Psi^i(p;q_1)$.

We now define the following random variable:
(4.2)

$$\boldsymbol{\delta}^i(p) = \begin{cases} \Pr_{\mathbf{q}_1,\mathbf{q}_2}\left[\tau^i(p;\mathbf{q}_1)\le\frac{1}{2} \ \wedge\ \mathcal{A}^i(p;\mathbf{q}_1,\mathbf{q}_2)\neq\dagger \mid \mathbf{i}=i\right], & \text{if } \mathbf{r}^i=1 \\ \Pr_{\mathbf{q}_2}\left[\mathcal{A}^i(p;\mathbf{Q}^i,\mathbf{q}_2)\in T^i(p;\mathbf{Q}^i)\mid \mathbf{i}=i\right], & \text{if } \mathbf{r}^i=2 \end{cases}$$

Note that this is a random variable, since it depends on $\mathbf{Q}$ and $\mathbf{r}$. However, also notice that $\mathbf{d}$ is irrelevant. Indeed, we are playing a formal game of feeding p to the query algorithm as the published bits. The value p may be totally unrelated to what the data structure could actually want to publish, i.e. $\mathbf{p}(\mathbf{r},\mathbf{Q},\mathbf{d})$.

Our analysis will actually concentrate on the function $\delta^i(p) = \mathbf{E}_{\mathbf{r},\mathbf{Q}}[\boldsymbol{\delta}^i(p)]$. Notice that this is no longer a random variable. For convenience, we also define $\delta^*(p) = \mathbf{E}_{i\in[k]}[\delta^i(p)] = \frac{1}{k}\sum_i\delta^i(p)$. Using standard notation from probability theory, we write $\delta^i(p\mid E) = \mathbf{E}_{\mathbf{r},\mathbf{Q}}[\boldsymbol{\delta}^i(p)\mid E]$, for an event E. It is important to point out that T^i and τ^i are not affected by such a conditioning, since they are simply mathematical functions (although they are themselves defined with reference to the marginal distribution on Q). We also write $\delta^i(p\mid X)$ when we condition on some random variable X, i.e. $\delta^i(p\mid X)$ is a function $x\mapsto\delta^i(p\mid X=x)$.

To analyze the accept probability α, define $\alpha^i(p) = \Pr_{\mathbf{q}_1,\mathbf{q}_2}[\mathcal{A}^i(p;\mathbf{q}_1,\mathbf{q}_2)\neq\dagger\mid\mathbf{i}=i]$ and $\alpha^i(p;q_1) = \Pr_{\mathbf{q}_2}[\mathcal{A}^i(p;q_1,\mathbf{q}_2)\neq\dagger\mid\mathbf{i}=i,\mathbf{q}_1=q_1]$. We also define $\alpha^*(p)$ and conditioned versions similar to δ^i. Now $\alpha = \mathbf{E}_{\mathbf{r},\mathbf{Q},\mathbf{d}}[\alpha^*(\mathbf{p}(\mathbf{r},\mathbf{Q},\mathbf{d})\mid\mathbf{r},\mathbf{Q},\mathbf{d})]$.

Finally, we need definitions giving us a handle on error. Let $e(r,Q,d;q_1,q_2)$ be an indicator for whether the data structure makes an error when faced with the input (r,Q,d) on the data structure's side, and (q_1,q_2) on the query side. Now condition on $\mathbf{p}(\mathbf{r},\mathbf{Q},\mathbf{d})=p$, $\mathbf{i}=i$, $\mathbf{q}_1=q_1$ and $\mathbf{q}_2=q_2$. Depending on $\mathbf{r},\mathbf{Q}$ and $\mathbf{d}$, the algorithm may reject, make an error or compute the correct answer. Assuming the algorithm accepts the query (which depends entirely on p,i,q_1,q_2), let $\varepsilon^i(p;q_1,q_2)$ be the probability of making an error. Formally, $\varepsilon^i(p;q_1,q_2) = \mathbf{E}[e(\mathbf{r},\mathbf{Q},\mathbf{d};q_1,q_2)\mid\mathbf{p}(\mathbf{r},\mathbf{Q},\mathbf{d})=p,\mathbf{i}=i,\mathbf{q}_1=q_1,\mathbf{q}_2=q_2]$.

Note that the parameter p in ε^i has a vastly different significance as the p in δ^i. In ε^i, we condition on p being the bits published by the data structure, thus changing the probability space (restricting the input). In δ^i, we work in the whole probability space, but play the formal game of always feeding the query algorithm p as the published bits.

We write $\varepsilon^i(p) = \mathbf{E}_{\mathbf{q}_1,\mathbf{q}_2}[\varepsilon^i(p;\mathbf{q}_1,\mathbf{q}_2)\mid\mathbf{p}(\mathbf{r},\mathbf{Q},\mathbf{d})=p,\ \mathbf{i}=i]$. We also define $\varepsilon^*(p)$ and the conditioned versions in the same manner as for δ^i. We are now ready to state a key property of any solution for $\bigoplus^k f^{(2)}$:

Lemma 4.1. *There exist $\mathfrak{r}$ and $\mathfrak{Q}$, such that $\mathbf{E}_{\mathbf{d}}[\delta^*(\mathbf{p}(\mathfrak{r},\mathfrak{Q},\mathbf{d})\mid\mathbf{r}=\mathfrak{r},\ \mathbf{Q}=\mathfrak{Q},\ \mathbf{d})]\ge\frac{\alpha\varepsilon_\dagger}{64}$ and $\mathbf{E}_{\mathbf{d}}[\varepsilon^*(\mathbf{p}(\mathfrak{r},\mathfrak{Q},\mathbf{d})\mid\mathbf{r}=\mathfrak{r},\ \mathbf{Q}=\mathfrak{Q},\ \mathbf{d})]\le\varepsilon+\frac{\varepsilon_\dagger}{8}$.*

4.2 An Analysis of $\bigoplus^k f^{(2)}$. This section is dedicated to proving Lemma 4.1. First we show:

Lemma 4.2. *For any i and p, we have $\delta^i(p)\ge\alpha^i(p)/4$.*

Proof.

$$\begin{aligned}
\delta^i(p) &= \mathbf{E}_{\mathbf{r},\mathbf{Q}}[\boldsymbol{\delta}^i(p)] \\
&= \frac{1}{2}\mathbf{E}[\boldsymbol{\delta}^i(p)\mid\mathbf{r}^i=1]+\frac{1}{2}\mathbf{E}_{\mathbf{Q}^i}[\boldsymbol{\delta}^i(p)\mid\mathbf{r}^i=2] \\
&= \frac{1}{2}\sum_{q_1:\tau^i(p;q_1)\le1/2}\Pr[\mathbf{q}_1=q_1]\cdot\alpha^i(p;q_1) \\
&\qquad+\frac{1}{2}\sum_{q_1}\Pr[\mathbf{q}_1=q_1]\cdot\tau^i(p;q_1)\cdot\alpha^i(p;q_1) \\
&\ge \frac{1}{2}\sum_{q_1:\tau^i(p;q_1)\le1/2}\Pr[\mathbf{q}_1=q_1]\cdot\alpha^i(p;q_1) \\
&\qquad+\frac{1}{2}\sum_{q_1:\tau^i(p;q_1)\ge1/2}\Pr[\mathbf{q}_1=q_1]\cdot\frac{1}{2}\alpha^i(p;q_1) \\
&\ge \frac{1}{2}\sum_{q_1}\Pr[\mathbf{q}_1=q_1]\cdot\frac{1}{2}\alpha^i(p;q_1)=\frac{1}{4}\alpha^i(p). \qquad\square
\end{aligned}$$

The result of this lemma is in some sense ignoring the help given by the published bits, by assuming they are constantly set to some value p. Fortunately, we can show that the published bits cannot significantly alter the average behavior, since they are much fewer than the number of subproblems. The following lemmas are slight variants of lemmas used in our deterministic lower bound. They are essentially Chernoff bounds exploiting independence of δ^i and α^i over various i's:

Lemma 4.3. *With probability at least $1-\frac{(\alpha\varepsilon_\dagger)^2}{512}$ over random $\mathbf{r},\mathbf{Q}$ and $\mathbf{d}$, we have:*

$$(\forall)p:\ \delta^*(p\mid\mathbf{r},\mathbf{Q},\mathbf{d})\ge\frac{\alpha^*(p)}{4}-\frac{\alpha\varepsilon_\dagger}{256}$$

LEMMA 4.4. *With probability at least* $1 - \frac{(\alpha\varepsilon_\uparrow)^2}{512}$ *over random* $\mathbf{r}, \mathbf{Q}$ *and* $\mathbf{d}$*, we have:*

$$(\forall)p: \ \alpha^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \le \alpha^*(p) + \frac{\alpha\varepsilon_\uparrow}{256}$$

We combine Lemmas 4.3 and 4.4 by a union bound. We conclude that with probability at least $1 - (\alpha\varepsilon_\uparrow)^2/256$ over random $\mathbf{r}, \mathbf{Q}$ and $\mathbf{d}$, we have that $(\forall)p$:

$$\left.\begin{array}{l} \delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \ge \frac{\alpha^*(p)}{4} - \frac{\alpha\varepsilon_\uparrow}{256} \\ \alpha^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \le \alpha^*(p) + \frac{\alpha\varepsilon_\uparrow}{256} \end{array}\right\} \Rightarrow$$

$$\Rightarrow \delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) - \frac{\alpha^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})}{4} \ge -\frac{\alpha\varepsilon_\uparrow}{128}$$

Since this holds for all p, it also holds for $p = \mathbf{p}(\mathbf{r}, \mathbf{Q}, \mathbf{d})$, the actual bits that the data structure wants to publish. This means:

(4.3)
$$\Pr_{\mathbf{r},\mathbf{Q},\mathbf{d}}\left[\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) - \frac{\alpha^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})}{4} < -\frac{\alpha\varepsilon_\uparrow}{128}\right] \le \frac{(\alpha\varepsilon_\uparrow)^2}{256}$$

Remember that our lemma must simultaneously guarantee a large enough δ^*, and a small enough increase in error. We now deal with the increase in error. First define the random variable $\boldsymbol{\mathcal{A}} = \mathcal{A}^{\mathbf{i}}(\mathbf{p}(r, Q, \mathbf{d}); \mathbf{q}_1, \mathbf{q}_2)$. Now let $\phi(r, Q) = \mathbf{E}[e(r, Q, \mathbf{d}; \mathbf{q}_1, \mathbf{q}_2) \mid \mathbf{r} = r, \mathbf{Q} = q, \boldsymbol{\mathcal{A}} \ne \dagger]$. In other words, $\phi(r, Q)$ is the probability the algorithm makes an error, conditioned on the query being accepted and the choices of $\mathbf{r}$ and $\mathbf{Q}$. By definition of ε, $\mathbf{E}[\phi(\mathbf{r}, \mathbf{Q}) \mid \boldsymbol{\mathcal{A}} \ne \dagger] \le \varepsilon$. Now by Markov, we have $\Pr[\phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \varepsilon_\uparrow/8 \mid \boldsymbol{\mathcal{A}} \ne \dagger] \ge \varepsilon_\uparrow/8$. Remember that $\Pr[\boldsymbol{\mathcal{A}} \ne \dagger] \ge \alpha$. Then $\Pr[\phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \varepsilon_\uparrow/8 \text{ and } \boldsymbol{\mathcal{A}} \ne \dagger] \ge \alpha\varepsilon_\uparrow/8$. In particular, $\Pr[\phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \varepsilon_\uparrow/8] \ge \alpha\varepsilon_\uparrow/8$.

Now let us work in the probability space where $\phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \varepsilon_\uparrow/8$ holds. The probability of (4.3) cannot increase too much in this space:

$$(4.4) \quad \Pr_{\mathbf{r},\mathbf{Q},\mathbf{d}}\left[\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) - \frac{\alpha^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})}{4} < -\frac{\alpha}{16}\right.$$
$$\left.\mid \phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \frac{\varepsilon_\uparrow}{8}\right] \le \frac{(\alpha\varepsilon_\uparrow)^2/256}{\alpha\varepsilon_\uparrow/8} = \frac{\alpha\varepsilon_\uparrow}{32}$$

We now want to lower bound $\mathbf{E}[\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \mid \phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \varepsilon_\uparrow/8]$. Because $\delta^*, \alpha^* \in [0, 1]$, we have $\delta^*(\cdot) - \alpha^*(\cdot)/4 \ge -1/4$. We use this as a pessimistic estimate for the cases covered by (4.4). We obtain:

$$\mathbf{E}\Big[\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) - \frac{\alpha^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})}{4}$$
$$\mid \phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \frac{\varepsilon_\uparrow}{8}\Big] \ge -\frac{\alpha\varepsilon_\uparrow}{128} + \frac{\alpha\varepsilon_\uparrow}{32} \cdot \left(-\frac{1}{4}\right)$$
$$\Rightarrow \quad \mathbf{E}\left[\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \mid \phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \frac{\varepsilon_\uparrow}{8}\right]$$
$$\ge \quad \frac{1}{4}\mathbf{E}\left[\alpha^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \mid \phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \frac{\varepsilon_\uparrow}{8}\right] - \frac{\alpha\varepsilon_\uparrow}{64}$$

Finally, we must lower bound the expectation on the right-hand side. We have:

$$\mathbf{E}\left[\alpha^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \mid \phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \frac{\varepsilon_\uparrow}{8}\right]$$
$$= \quad \Pr\left[\boldsymbol{\mathcal{A}} \ne \dagger \mid \phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \frac{\varepsilon_\uparrow}{8}\right]$$
$$\ge \quad \Pr\left[\boldsymbol{\mathcal{A}} \ne \dagger \text{ and } \phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \frac{\varepsilon_\uparrow}{8}\right] \ge \frac{\alpha\varepsilon_\uparrow}{8}$$

This finally implies that:

$$\mathbf{E}\left[\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \mid \phi(\mathbf{r}, \mathbf{Q}) \le \varepsilon + \varepsilon_\uparrow/8\right] \ge \alpha\varepsilon_\uparrow/64$$

Now we use the probabilistic method. There exist $\mathfrak{r}$ and $\mathfrak{Q}$ such that $\phi(\mathfrak{r}, \mathfrak{Q}) \le \varepsilon + \varepsilon_\uparrow/8$, and $\delta^*(\mathbf{p} \mid \mathbf{r} = \mathfrak{r}, \mathbf{Q} = \mathfrak{Q}, \mathbf{d}) \ge \alpha\varepsilon_\uparrow/64$. These are exactly the conditions guaranteed by the lemma.

4.3 Constructing a Solution for $\bigoplus^k f$.

In this section, we complete the proof of Lemma 3.1.

We try to use the solution for $\bigoplus^k f^{(2)}$ to construct a better solution for $\bigoplus^k f$. To use this strategy, we need to extend an instance of $\bigoplus^k f$ to an instance of $\bigoplus^k f^{(2)}$. This is done using the $\mathfrak{r}$ and $\mathfrak{Q}$ values whose existence is guaranteed by Lemma 4.1. The extended data structure's input consists of the vector $(d^1, \dots, d^k)$ given to $\bigoplus^k f$, and the vectors $\mathfrak{r}$ and $\mathfrak{Q}$. A query's input for $\bigoplus^k f$ is a problem index $i \in [k]$ and a $q \in Q$. We extend this to $(\mathfrak{Q}^i, q)$ if $\mathfrak{r}^i = 2$, or manufacture an appropriate suffix if $\mathfrak{r}^i = 1$, as described below.

First note that extending an input of $\bigoplus^k f$ to an input of $\bigoplus^k f^{(2)}$ by this strategy preserves the desired answer to a query (in particular, the suffix is irrelevant to the answer, so it can be manufactured). Also, this transformation is well defined because $\mathfrak{r}$ and $\mathfrak{Q}$ are "constants", defined by the input distribution $\bigoplus^k \mathcal{D}^{(2)}$. Since our model is nonuniform, we only care about the existence of $\mathfrak{r}$ and $\mathfrak{Q}$, and not about computational aspects. From the point of view of the data structure, we are done: we simply construct the representation and the help bits by using the algorithm for $\bigoplus^k f^{(2)}$ on the extended input.

It remains to describe the new query algorithm. First, we consider a hybrid problem: $\bigoplus^k f^{(2)}$ given the choices of $\mathfrak{r}$ and $\mathfrak{Q}$. This is a special case of $\bigoplus^k f^{(2)}$ because we restrict the input through $\mathfrak{r}$ and $\mathfrak{Q}$. Also, it is not the same as the problem we are ultimately interested in, because when $\mathbf{r^i} = 1$, the suffix of the query is random, and not manufactured by the algorithm. In this hybrid problem, we transform all queries with a noticibly higher error rate into rejects (which we call *error rejects*). Specifically, if $\varepsilon^i(p; q_1, q_2 \mid \mathbf{r} = \mathfrak{r}, \mathbf{Q} = \mathfrak{Q}) > \varepsilon + \varepsilon_\uparrow$, the query algorithm rejects when getting query (q_1, q_2) for subproblem i, with the published bits being p. Note that this is well-defined, since ε^i is a mathematical function depending only on information known to the querier.

Intuitively, two things can happen after this transformation. If the error is well-spread across queries, the rejects introduced by this technique are not significant. Then, we can apply an algorithmic construction (see below) for reducing the space of initial probes, which uses the guarantee that no single accepted query is too bad in terms of error. Otherwise, a sizable fraction of the error is concentrated on a few queries. Then, we improve in another direction: we decrease the overall error, introducing just a few rejects for the bad queries.

Formally, let $\rho^i(p; q_1, q_2)$ be an indicator for error rejects (i.e. it is 1 in we reject the combination p, i, q_1, q_2 and 0 otherwise). To simplify notation, also let $\lhd(p)$ be the event characterized by $\mathbf{r} = \mathfrak{r}$, $\mathbf{Q} = \mathfrak{Q}$, and $\mathbf{p}(\mathfrak{r}, \mathfrak{Q}, \mathbf{d}) = p$. We now make the following definitions:

$\rho^i(p) = \mathbf{E}_{\mathbf{q}_1, \mathbf{q}_2}\big[\rho^i(p; \mathbf{q}_1, \mathbf{q}_2) \mid \lhd(p), \mathbf{i} = i\big]$.

$\rho^* = \mathbf{E}_{\mathbf{d}, \mathbf{i}}\big[\rho^{\mathbf{i}}(\mathbf{p}(\mathfrak{r}, \mathfrak{Q}, \mathbf{d}))\big]$.

$\rho^i(p; q_1) = \mathbf{E}_{\mathbf{q}_2}[\rho^i(p; q_1, \mathbf{q}_2)]$. This is only defined if $\mathfrak{r}^i = 1$. It is important to realize that the distribution of $\mathbf{q}_2$ is the marginal distribution on Q, even given $p, \mathfrak{r}$ and $\mathfrak{Q}$. Indeed, the data structure has no information about the suffix, since $\mathfrak{r}^i = 1$.

The two cases discussed above can be understood formally in terms of ρ^*.

The case $\rho^* \geq \frac{\alpha\varepsilon_\uparrow}{256}$. We aim for case 2 of the central lemma: reduce the error of the query algorithm, without changing the set of initial probes. For this, consider the instances of p, i, q_1, q_2 sorted by increasing $\varepsilon^i(p; q_1, q_2 \mid \mathbf{r} = \mathfrak{r}, \mathbf{Q} = \mathfrak{Q})$. Pick the best instances, until we reach a set of total probability mass roughly $\frac{\alpha\varepsilon_\uparrow}{128}$. A subset with mass negligibly close to this can be chosen because the granularity is small enough: each query has probability at most $1/k$ due to the uniform choice of the subproblem.

Now consider the algorithm in which we reject all but the instances chosen as above. Clearly, the overall accept probability is $\Omega(\alpha\varepsilon_\uparrow)$, as desired. Now we argue that the average error has gone down by $\Omega(\varepsilon_\uparrow)$. Indeed, δ^* is a lower bound for the accept probability, so by Lemma 4.1, the old algorithm accepted with probability at least $\frac{\alpha\varepsilon_\uparrow}{64}$. Then, a total mass of $\frac{\alpha\varepsilon_\uparrow}{128}$ was rejected. Among these instances, there were at least $\frac{\alpha\varepsilon_\uparrow}{256}$ which had an error rate of $\varepsilon + \varepsilon_\uparrow$. This is higher by $7\varepsilon_\uparrow/8$ compared to the overall error rate of $\varepsilon + \varepsilon_\uparrow/8$ (by Lemma 4.1). Excluding the error rejects makes the average error go down by at least $(1/4)(7\varepsilon_\uparrow/8) = 7\varepsilon_\uparrow/32$, and excluding other instances in decreasing order can only improve the average. Then the new average error rate is $\varepsilon + \varepsilon_\uparrow/8 - 7\varepsilon_\uparrow/32 = \varepsilon - 3\varepsilon_\uparrow/32$.

To simplify expressions, denote the new average error by $\varepsilon - 2c\varepsilon_\uparrow$, for $c = 3/64$. Ideally, we would like all instances to have small error, slightly above the average, e.g. $\varepsilon - c\varepsilon_\uparrow$. If most do, we are done (see below). Otherwise, we can apply the same reasoning as above: reject a constant fraction of the probability mass, but improve the error further. We repeat this an arbitrary number of times, until the termination condition is reached.

More precisely, after j repetitions (starting with $j = 0$), the average error will be $\varepsilon - (j+2)c\varepsilon_\uparrow$. Now consider all instances with error rate above $\varepsilon - (j+1)c\varepsilon_\uparrow$. If they have mass at most a half, we are done, and we continue as described below. Otherwise, we keep only the best instances in terms of error rate, up to a probability mass of a half. Thus, we halve the accept rate, but improve the average error rate by $c\varepsilon_\uparrow$. Now repeat. To finalize the construction, we reject all instances with error above $\varepsilon - (j+1)c\varepsilon_\uparrow$. These have mass at most a half, since the loop terminated. Thus, in the end the accept probability is $\Omega(\alpha/2^j)$.

Now we finally convert the algorithm for $\bigoplus^k f^{(2)}$ into an algorithm for $\bigoplus^k f$. When $\mathfrak{r}^i = 2$, the algorithm simply passes the query to the old algorithm. Thus, the accept and error rates for such a subproblem are unchanged. When $\mathfrak{r}^i = 1$, we can choose an arbitrary suffix, leading to an accepted query. The accept probability can only improve, since even one good suffix suffices to make the query accepted. Also note that the average error is bounded by $\varepsilon - (j+1)c\varepsilon_\uparrow$, since for *any* accepted suffix we have at most this error.

The case $\rho^* < \frac{\alpha\varepsilon_\uparrow}{256}$. We aim for the guarantees in case 1 of the central lemma: reduce the range of the first cell probe, sacrificing the other parameters. When our algorithm accepts a query, it will simulate a non-rejecting query in the hybrid algorithm for $\bigoplus^k f^{(2)}$. Since all queries with error above $\varepsilon + \varepsilon_\uparrow$ have been error-rejected, the average error of the algorithm can be at most $\varepsilon + \varepsilon_\uparrow$. Thus, we only need to analyze the accept probability as we develop the new algorithm. The following two lemmas describe the two strategies of the algorithm:

LEMMA 4.5. *For any instance p of the published bits and a subproblem i with $\mathfrak{r}_i = 2$, there is a query algorithm whose first probe is in $T^i(p; \mathfrak{Q}^i)$ when the query is accepted, and which has an accept probability of at least $\delta^i(p \mid \lhd(p)) - \rho^i(p)$.*

Proof. The algorithm probe-rejects all queries which lead to a probe outside $T^i(p; \mathfrak{Q}^i)$, and runs the others by the old algorithm. By definition, $\delta^i(p \mid \lhd(p)) = \Pr_{\mathbf{q}_2}[\mathcal{A}^i(p; \mathfrak{Q}^i, \mathbf{q}_2) \in T^i(p; \mathfrak{Q}^i) \mid \mathbf{i} = i \wedge \lhd(p)]$. Subtracting $\rho^i(p)$ eliminates those $\mathbf{q}_2$ values which are error rejected; possibly, some were not even counted by the previous expression, so this is a lower bound. Thus, we are left with queries which are not error-rejected, and want to probe a cell inside $T^i(p; \mathfrak{Q}^i)$. $\square$

LEMMA 4.6. *For any instance p of the published bits, there exists a set of cells H with $|H| = k\sqrt{\sigma} \cdot O(w)$, such that for every subproblem i with $\mathfrak{r}_i = 1$, there is a query algorithm whose first probe is in H whenever the query is accepted, and which has an accept probability of at least $\delta^i(p \mid \lhd(p)) - 2\rho^i(p)$.*

Proof. Given a set H, the query algorithm is simple to describe. We iterate through all possible query suffixes. For each one, we simulate the extended query using the algorithm for $\bigoplus^k f^{(2)}$. If this algorithm rejects the query, or the first probed cell is not in H, we continue trying suffixes. If we don't find any good suffix, we probe-reject the query. Note that it is essential that the old algorithm accepts or rejects after looking just at published bits. Then, searching for a suffix that would not be rejected is free, as it does not involve any cell probes. Also, it is essential that *any* extended query has a good error probability, since we do not know which suffix will be used.

It remains to justify the existence of a good enough set H. Let $\mathcal{C}$ be the algorithm after we have introduced the error rejects. Consider all subproblems i and queries q such that $\{\mathcal{C}^i(p; q, q_2) \mid q_2 \in Q\} \setminus \{\dagger\}$ has cardinality at least $\sqrt{\sigma}$. These are subsets of the universe $\mathcal{C}^*(p) \subseteq \mathcal{A}^*(p)$, and $|\mathcal{A}^*(p)| \leq k\sigma$ by assumption of Lemma 3.1. Then, by a standard probabilistic argument, a random subset of $\mathcal{A}^*(p)$ with density roughly $1/\sqrt{\sigma}$ will intersect all these sets with nonzero probability. Specifically, we have at most 2^b sets, because i and q are part of the query (which is a word), so there exists a subset H with $\Theta(k\sqrt{\sigma}b)$ elements which intersects all our sets.

Finally, we need to analyze the success probability of this algorithm. Consider some pair i and q with $\tau^i(p; q) \leq 1/2$. Our goal is to show that this pair can be handled by the above argument if $2\rho^i(p; q) \leq \alpha^i(p; q)$. Note that the right hand side counts the suffixes which were accepted before. Overall, the condition states the error rejects themselves decrease this accept probability by at most a half. As for $\rho^i(p; q)$, it is important to realize that $\alpha^i(p; q \mid \lhd p) = \alpha^i(p; q)$, so this quantity is relevant in the probability space we care about. Indeed, the distribution of $\mathbf{q}_2$ is in both cases the marginal distribution on Q, since the data structure has no information about $\mathbf{q}_2$.

To prove the claim, consider the definition of $\tau^i(p; q)$. This only relies on randomness over $\mathbf{q}_2$, so it is unchanged if we condition on $\lhd p$. Thus, $\tau^i(p; q)$ counts the probability mass of the most frequently-probed $\sqrt{\sigma}$ cells. By assumption, this is at most a half. Then, if an arbitrary set of queries with mass at most a half is error-rejected, one still cannot reduce the number of cells that are probed by accepting queries to less than $\sqrt{\sigma}$. Then, as explained above, the set H can be chosen to satisfy at least one accepting suffix.

We finally analyze the probability that $\mathbf{q}$ will satisfy $\tau^i(p; \mathbf{q}) \leq 1/2$ and $2\rho^i(p; \mathbf{q}) \leq \alpha^i(p; \mathbf{q})$. We use $\mathbf{Ind}[\cdot]$ for the indicator of a condition (1 if the condition holds, 0 otherwise). Observe that for any variable $x \leq 1$, $\mathbf{Ind}[x \geq 0] \geq x$.

$$
\begin{aligned}
&\Pr_{\mathbf{q}}\left[\tau^i(p; \mathbf{q}) \leq 1/2 \ \wedge \ 2\rho^i(p; \mathbf{q}) \leq \alpha^i(p; \mathbf{q}) \mid \mathbf{i} = i \wedge \lhd(p)\right] \\
&= \sum_{q : \tau^i(p;q) \leq 1/2} \Pr[\mathbf{q} = q \mid \mathbf{i} = i \wedge \lhd(p)] \ \cdot \\
&\qquad\qquad \cdot \mathbf{Ind}[\alpha^i(p; q) - 2\rho^i(p; q) \geq 0] \\
&\geq \sum_{q : \tau^i(p;q) \leq 1/2} \Pr[\mathbf{q} = q \mid \mathbf{i} = i \wedge \lhd(p)] \cdot (\alpha^i(p; q) - 2\rho^i(p; q)) \\
&\geq \sum_{q : \tau^i(p;q) \leq 1/2} \Pr[\mathbf{q} = q \mid \mathbf{i} = i \wedge \lhd(p)] \cdot \alpha^i(p; q) \\
&\qquad - \sum_{q} \Pr[\mathbf{q} = q \mid \mathbf{i} = i \wedge \lhd(p)] \cdot 2\rho^i(p; q) \\
&= \ \mathbf{E}_{\mathbf{q}}\left[\alpha^i(p; \mathbf{q}) \cdot \mathbf{Ind}[\tau^i(p; \mathbf{q}_1) \leq 1/2] \mid \mathbf{i} = i \wedge \lhd(p)\right] \\
&\qquad -2\,\mathbf{E}_{\mathbf{q}}[\rho^i(p; \mathbf{q}) \mid \mathbf{i} = i \wedge \lhd(p)] \\
&= \ \delta^i(p \mid \lhd(p)) - 2\rho^i(p). \qquad \square
\end{aligned}
$$

Thus, for every choice of the published bits, our algorithm only probes cells from a set of size $k \cdot \sqrt{\sigma} + k\sqrt{\sigma} \cdot O(w)$, as desired. Furthermore, the average accept probability over all subproblems and all choices of p is at least $\mathbf{E}_{\mathbf{d}}[\delta^*(\mathbf{p}(\mathfrak{r}, \mathfrak{Q}, \mathbf{d}) \mid \mathbf{r} = \mathfrak{r}, \mathbf{Q} = \mathfrak{Q}, \mathbf{d})] - 2\rho^*$. By Lemma 4.1, this is at least $\frac{\alpha\varepsilon_\uparrow}{64} - 2\frac{\alpha\varepsilon_\uparrow}{256} = \Omega(\alpha\varepsilon_\uparrow)$.

References

[1] M. Ajtai. A lower bound for finding predecessors in Yao's cell probe model. *Combinatorica*, 8(3):235–247, 1988.

[2] S. Alstrup, G. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000.

[3] P. Beame and F. E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002. See also STOC'99.

[4] A. Chakrabarti and O. Regev. An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In *Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 473–482, 2004.

[5] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small forwarding tables for fast routing lookups. In *Proc. ACM SIGCOMM*, pages 3–14, 1997.

[6] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *Proc. IEEE INFOCOM*, pages 1193–1202, 2000.

[7] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993. See also STOC'90.

[8] D. Liu. A strong lower bound for approximate nearest neighbor searching. *Information Processing Letters*, 92(1):23–29, 2004.

[9] P. B. Miltersen. Lower bounds for Union-Split-Find related problems on random access machines. In *26th ACM Symposium on Theory of Computing (STOC)*, pages 625–634, 1994.

[10] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998. See also STOC'95.

[11] M. Pătraşcu and M. Thorup. Higher lower bounds for near-neighbor and further rich problems. In *Proc. 47th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

[12] M. Pătraşcu and M. Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.

[13] P. Sen and S. Venkatesh. Lower bounds for predecessor searching in the cell probe model. *arXiv:cs.CC/0309033*. See also ICALP'01, CCC'03, 2003.

[14] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977. Announced by van Emde Boas alone at FOCS'75.

[15] D. E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983.

[16] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.