10

Lower Bounds for Summaries

The focus of this volume so far has been on what is possible to effectively summarize. Yet it should come as no surprise that there are some questions for which no summary exists. In this chapter, we look at cases where it is not simply that no summary is known to exist, but rather where it is mathematically impossible for any summary to be made that is less than some size. The intuition underlying these results are that certain settings require a large amount of information to be stored in order to allow a range of queries to be answered accurately. Where this information is comparable to the size of the data being stored, it effectively precludes the possibility of a useful summary for that problem. In other cases, summaries are possible, and the lower bounds tell us how small we can hope the summary to be.

Computational complexity is the area of computer science devoted to understanding the fundamental limits of what can be computed effectively. Most Computer Science degrees cover the time complexity of solving problems based on classes such as $\mathcal{P}$ and $\mathcal{NP}$. $\mathcal{P}$ is the class of problems that can be solved in a standard model of computation in time that is bounded by a polynomial in the input size, denoted n. $\mathcal{NP}$ is the class of problems (informally) where a conjectured solution can be *verified* in time polynomial in n. The notion of *hardness* in this setting is to show that a given problem is as hard as another, in the sense that a polynomial time algorithm for the former problem would yield a polynomial time algorithm for the latter.

These notions of complexity do not translate to the world of summaries for a number of reasons. Primarily, summaries are often relevant when bounds that are polynomial would be considered too lax. For the most part, the summaries described in this book address problems which can be easily solved in time polynomial in the input size n.

Rather, we look for summaries which use time and space resources that are smaller than n, ideally strictly sublinear in n.

Techniques that provide lower bounds for summaries tend to be derived from the area of communication complexity. These study the cost of communication between two parties (traditionally, anthropomorphized as Alice and Bob) who each hold part of the input, and wish to collaborate to compute a function of it. For example, suppose Alice and Bob hold strings x and y, and wish to compute whether $x = y$ or $x \neq y$. We refer to this as the EQUALITY problem. In this setting, a trivial protocol is for Alice to send her part of the input to Bob, which would take $O(n)$ communication. Therefore, the way to show bounds on summary size is to derive bounds on communication, up to linear in n.

There is a very natural mapping from bounds on communication to bounds on summaries. We can view the communication between Alice and Bob as a communication from the past to the present. That is, suppose we had a summary that addressed a particular problem that maps onto a particular communication problem. We could have Alice run the summary algorithm on her portion of the input to build the summary data structure in memory. This could then be communicated to Bob, who could subsequently put his portion of the input into the summary, either by performing a MERGE with a summary of his data, or applying repeated UPDATE operations. If the summary provides an answer via a QUERY operation which can be interpreted as an answer to the communication problem, then we have a communication protocol.

Therefore, any lower bound on the communication needed to solve the communication problem provides a corresponding lower bound on the size of a summary to solve the relevant summarization problem. We will see several examples of this outline being instantiated over the course of this chapter. The application of data summarization (particularly in the context of streaming data processing) has stimulated the area of communication complexity, and led to many novel techniques being developed in order to prove new lower bounds for summaries. The lower bounds tend to be on the size of the summary, rather than on the time necessary to UPDATE or QUERY them: this is a result of the focus on communication size from communication complexity. In other words, the techniques we have for proving hardness most naturally provide lower bounds on summary size; proving bounds on the time costs associated with data summarization would also be very important, but has proved to be more challenging with the tools currently available.

We will not provide proofs of the communication complexity lower bounds, which warrant a volume of their own to define (as a starting point, see the text of Kushilevitz and Nisan [157]). Rather, we focus on the ways that hard communication problems provide lower bounds on summary sizes. These tend to be *reductions*: for a given summary problem, we identify a suitable communication problem that would be solved if a small summary existed. For the remainder of this chapter, we describe some of the commonly used hard problems in communication complexity, and give examples of the summaries for which they provide lower bounds. An important concept is whether the communication bounds are one round (for protocols where Alice sends a single message to Bob), or multi-round (the bounds still hold when Alice and Bob are allowed to have a conversation with many messages back and forth).

10.1 Equality and Fingerprinting

The most basic communication complexity problem is the EQUALITY problem. Here, Alice and Bob both possess binary strings of length n, denoted as x and y respectively. The problem is to determine whether or not $x = y$. It is straightforward to see that if Alice is to send a single message to Bob, then it must contain n bits, if Bob is to be guaranteed to give the correct answer. Suppose the contrary: Alice sends a message of $b < n$ bits in length. Alice has 2^n possible inputs, but only $2^b < 2^n$ possible messages to choose from. So by the pigeonhole principle, there must be two different inputs x and x' that Alice could have which would cause her to send the same message to Bob. Then Bob cannot guarantee to succeed, as he may have $y = x$ as his string, and be uncertain whether Alice held x (requiring a 'yes' answer) or x' (requiring a 'no' answer). Therefore the problem must require a communication of at least n bits. Applying the above template, this also means that any summary that claims to answer the corresponding problem — summarizing two inputs to determine whether or not they are equal — must also require $\Omega(n)$ bits

Connection to Fingerprinting. On first glance, this hardness may appear to cause a contradiction. We have studied a summary (Fingerprint) which claims to solve exactly this problem using a summary size that is much smaller than n bits. This apparent contradiction is resolved by observ-

ing that the argument above required Alice and Bob to be deterministic in their operation. That is, we have successfully shown that any *deterministic* summary must be linear in the input size if it allows equality to be tested and verified with certainty. However, if Alice and Bob are allowed to make random choices, and tolerate some small probability of error, then a much smaller summary is possible, as witnessed by the Fingerprint method. This serves to highlight the importance of randomization: for the majority of problems we consider, randomization is required to evade strong lower bounds on deterministic summaries. The subsequent examples we consider provide lower bounds on communication schemes which allow randomization, which therefore provide lower bounds on randomized summary techniques.

10.2 Index and Set Storage

The INDEX problem in communication complexity is defined as follows:

Definition 10.1 (INDEX problem) *Alice holds a binary string x of n bits in length, while Bob holds an index y in $[n]$. The goal is for the players to follow a protocol to output $x[y]$, i.e., the y'th bit from the string x.*

If the players are allowed multiple rounds of communication between them, then there is a trivial protocol for INDEX: Bob communicates y to Alice, who then emits $x[y]$, requiring $\lceil \log n \rceil + 1$ total bits of communication. However, under the constraint that Bob does not communicate to Alice, the problem becomes much harder. That is, in the 'one-way' communication complexity model, Alice sends a message to Bob, who must then output the answer. It is hard to imagine that one can find any solution appreciably better than Alice sending her entire string to Bob, who can then read off the required bit. Indeed, it is straightforward to adapt the above proof for EQUALITY to show that no deterministic algorithm for this problem can communicate fewer than $\Omega(n)$ bits of information.

It is more involved to show that this remains the case when Alice and Bob are allowed to use randomization. Intuitively, it is not clear how randomization would help here, but a formal lower bound must rule out the possibility of some clever summary that encodes Alice's string in some non-obvious way. Such a proof was first shown in the early 1990s by Razborov [197]. A compact proof of this fact which uses

the method of information complexity is provided by Jayram [137]. Formally, these results show that even when the randomized communication protocol is required to succeed only with some constant probability (say, $\frac{2}{3}$), the players still need to communicate $\Omega(n)$ bits.

It is natural to apply this result to summary techniques. The constraint that Bob is not allowed to communicate his index y to Alice is a natural one: we can think of Alice as observing the data, from which she must construct her summary (message). Bob then receives the summary and wants to answer some query. However, if the answer to Bob's query could reveal the value of any one of n bits encoded into the summary by Alice, then we know that the summary must have size $\Omega(n)$.

Lower bound for Set Storage. A direct application of this is to show a size bound for any summary that encodes a set, such as a BloomFilter (Section 2.7). The argument proceeds as follows. Suppose we had a summary that could very compactly encode a set A of items from n possibilities. Now, take an instance of INDEX. Alice could store her bitstring in the set summary, by storing each element i such that $x[i] = 1$. After the summary is sent to Bob, Bob looks up y in the summary. If y is present in the set, then Bob asserts that $x[y] = 1$, else he concludes $x[y] = 0$. Therefore, any summary that encodes a set in this way would provide a solution to the INDEX problem, and hence must use at least $\Omega(n)$ bits. In particular, this argument shows that the BloomFilter must use $\Omega(n)$ bits to represent sets whose size is proportional to n.

Lower bound for Count Distinct. A slightly more elaborate argument can show a lower bound on the size of summaries to estimate the number of distinct members of a set, such as KMV and HLL. Consider again an instance of the INDEX problem, and a summary which claims to allow the estimation of the number of distinct elements in a set, up to an approximation factor of $1 \pm \epsilon$. Alice performs a similar reduction to above: she performs an UPDATE operation on the distinct elements summary with each index i such that $x[i] = 1$. On receiving the summary, Bob first performs a QUERY operation to estimate the size of Alice's set, as m. Bob then performs an UPDATE with his index y, and makes a second QUERY to the summary, to obtain a second estimate m'. These two answers are compared as follows: if $(1 - \epsilon)m' > (1 + \epsilon)m$, then Bob concludes that the number of distinct elements has increased following the insertion of y. That is, y was not previously in the set, and so $x[y] = 0$; otherwise, Bob concludes that $x[y] = 1$. This argument works if the parameter ϵ is small enough so that a change of 1 in the size of the sets must lead to a

different answer. That is, $\epsilon n < 1$, for a set size of n. The lower bound of $\Omega(n)$ then implies a lower bound in terms of ϵ as $\Omega(1/\epsilon)$ on the size of the summary.

Observe that this lower bound is somewhat unsatisfying, since both KMV and HLL are shown to have a space cost that depends on $O(1/\epsilon^2)$. A reduction to a stronger lower bound below (Section 10.4) removes this gap.

Lower bound for Counting Triangles. The hardness of INDEX can also be used to show the hardness of the problem of counting triangles in a graph. Upper bounds for this problem are discussed in Section 7.4. Here, we formalize the problem of counting triangles as a problem parametrized by a scalar T. We are asked to distinguish between graphs that have no triangles at all, and those which have at least $T < n$ triangles. Given a summary technique which claims to solve this problem, we show how to use it to solve INDEX. We construct the graph over three sets, X, Y, and Z, where Z is chosen to be of size T. The bitstring x is encoded into the adjacency pattern between X and Y: index i is mapped in some canonical fashion to a pair of indices j and k, and edge (X_j, Y_k) is placed in the graph by Alice if the corresponding bit $x_i = 1$. This encoding allows Bob to probe for the value of a bit in x, as follows. Bob similarly finds the indices j and k from his index y, and inserts edges (X_j, Z_ℓ) and (Y_k, Z_ℓ) for all $1 \leq \ell \leq T$. Then, if there is the edge (X_j, Y_k) in the graph (corresponding to $x_i = 1$), there are a total of T triangles, whereas there are no triangles otherwise. This shows that the triangle counting problem requires space $\Omega(|X| \cdot |Y|)$. A nice feature of this reduction (due to Braverman, Ostrovsky and Vilenchik [37]) is that the sizes of X and Y can be chosen to generate an arbitrary number of edges as a function of n, so the problem is hard, whether the graph is sparse (has only $O(n)$ edges) or dense (has $\Omega(n^2)$ edges), or anywhere in between. The number of edges, m, is $O(nT)$, for an INDEX instance of size n, so the summary size lower bound in terms of m is $\Omega(m/T)$.

10.3 Disjointness and Heavy Hitters

The DISJOINTNESS problem is defined as follows:

Definition 10.2 (DISJOINTNESS problem) *Alice holds a binary string x of n bits in length, while Bob holds a binary string y, also of n bits in length. The*

goal is for the players to determine whether there exists an index i such that $x[i] = y[i] = 1$, or that no such index exists.

The problem may appear tractable, but it is hard. Formally, any communication protocol for DISJOINTNESS requires the players to communicate a total of $\Omega(n)$ bits, even if randomization is allowed. This holds even when the players are allowed to have multiple rounds of interaction: Alice and Bob can send multiple messages. This relaxation does not help in proving stronger lower bounds for summary construction, but is useful for places where multiple rounds of communication could be allowed, such as in the distributed setting (Chapter 8). The hardness of the DISJOINTNESS problem is shown by a similar argument. It was first demonstrated by Kalyanasundaram and Schnitger [145], and simplified by Razborov [197].

Note that if we change the problem to ask whether there exists any index i such that $x[i] = y[i]$ (i.e., we remove the requirement that the bit at the index is 1), then the new problem becomes much simpler. If we have a 'no' instance of this new problem then we must have $x[i] \neq y[i]$ for all locations i. That is, $x[i] = (1 - y[i])$. Then we can compare fingerprints of x and the string y' formed by flipping every bit of y, and output 'no' if these fingerprints match. This protocol has communication cost $O(\log n)$. Thus, the asymmetry in the problem definition is required to make it a difficult problem.

Lower bound for Multiset frequency. A first application of the DISJOINTNESS problem is to show the hardness of estimating the highest frequency in a multiset. That is, the input defines a multiset v, and the objective is to estimate $\max_i v_i$, the largest frequency in the multiset. Again, assume we had a compact summary for this problem, and we will show how it could be used to solve DISJOINTNESS. Given the instance of DISJOINTNESS, Alice takes her string x, and encodes it so that $v_i = x[i]$, i.e., inserts i into the summary if $x[i] = 1$. This summary is sent to Bob, who similarly encodes his string. This means that $v_i = x[i] + y[i]$. Suppose we could find $F = \max_i v_i$. Then $F = 2$ if and only iff there is some i such that $x[i] = y[i] = 1$, and $F \leq 1$ otherwise. Hence, we can't hope to solve this maximum frequency problem with a summary of size less than $\Omega(n)$, even if we allow an approximate answer[1]. This explains why the various summaries that address this problem (MG, SpaceSaving,

[1] That is, even if we only approximate F up to a constant factor less than 2.

Count-Min Sketch and Count Sketch) offer a different guarantee: they approximate frequencies not with relative error, but with an error that depends on $\|v\|_1$ or $\|v\|_2$.

Lower bound for inner product estimation. Section 6.1 shows how sketches can be used to estimate inner-products between pairs of vectors u and v, with an error that is proportional to $\|u\|_2\|v\|_2$. In many applications, we would prefer to have error that scales proportional to the inner product $u \cdot v$ itself. This is not feasible in general for arbitrary inputs, by a reduction to the DISJOINTNESS problem.

Suppose that we had a summary that promised to answer inner product queries with error $\epsilon(u \cdot v)$. Then we could use it to solve DISJOINTNESS quite directly: simply set $u = x$ and $v = y$ for binary vectors x and y. Then observe that $(x \cdot y) = 0$ iff x and y are disjoint (a no instance of the problem), but $(x \cdot y) \geq 1$ iff x and y have any point of intersection. That is, the result counts the number of points of intersection. Then the space required by any summary must be $\Omega(n)$, else it could be used as the basis of a DISJOINTNESS communication protocol. In particular, this rules out any constant factor approximation of $(x \cdot y)$, since this would allow us to distinguish between the $x \cdot y = 0$ and $x \cdot y \geq 1$.

This argument also allows us to argue that error terms like $\epsilon\|u\|_2\|v\|_2$ are reasonable. Consider the same interpretation of bitstrings from DISJOINTNESS as vectors. Then $\|u\|_2, \|v\|_2 = \Omega(\sqrt{n})$ in general, where n denotes the length of the bitstrings. Estimating their inner-product with sufficient accuracy to distinguish a 0 from a 1 result would require the additive error to satisfy $\epsilon\|u\|_2\|v\|_2 = \epsilon\Omega(n) < \frac{1}{2}$, i.e., $\epsilon = O(1/n)$, or $n = \Omega(1/\epsilon)$. This implies that the size of the summary must be $\Omega(1/\epsilon)$.

Multi-pass lower bound for triangle counting. In Section 10.2, we used the hardness of the INDEX problem to show that counting the number of triangles in a graph requires a summary of size proportional to the number of edges. Here, we describe a stronger lower bound [65], under the more demanding scenario that we are allowed to access the input data multiple times. This corresponds to a communication problem where Alice and Bob can exchange multiple messages. Note that INDEX is not helpful to us here: if Alice and Bob can have multiple rounds of communication, then we can easily solve INDEX instances with few bits: Bob just has to send the index to Alice, who can reply with the target value. Instead,

we rely on problems like DISJOINTNESS, which remain hard even when multiple rounds of communication are allowed.

As before, we focus on the problem of distinguishing a graph G with no triangles from one with T or more triangles. We will construct a graph on $\theta(n)$ vertices with $\Omega(n\sqrt{T})$ edges and $1 \leq T \leq n^2$ triangles by encoding an instance of DISJOINTNESS. We define three sets of nodes, A with n nodes, and B and C which each have $\sqrt{T}$ nodes. Given the binary string y over n bits, Bob creates edges from each a_i such that $y_i = 1$ to all nodes in B. Alice creates every edge (b, c) between $b \in B$ and $c \in C$. Last, she inserts edges from each a_i such that $x_i = 1$ to all nodes in C.

Observe from this construction that if strings x and y are disjoint, then there are no triangles in the graph, since there is no node a_i with an edge to B and an edge to C. However, for every intersection i between x and y there are triangles on nodes (a_i, b_j, c_k) for all $\sqrt{T}$ values of j and k, giving a total of T triangles. The bound $T \leq n^2$ ensures that the number of nodes is $O(n)$, and the number of edges is $T + (|x| + |y|)\sqrt{T} \leq 3n\sqrt{T}$ (using $\sqrt{T} \leq n$ from the assumption).

From the hardness of DISJOINTNESS, we know that any summary for this problem must use $\Omega(n)$ space. Rewriting the bound in terms of the number of edges, m, we obtain a bound of $\Omega(m/\sqrt{T})$. This is stronger than the corresponding bound of $\Omega(m/T)$ by using a reduction from INDEX.

10.4 Gap Hamming and Count Distinct, Again

To prove stronger bounds for some problems, a different problem is defined, based on the notion of Hamming distance: the number of places where a pair of binary strings differ.

Definition 10.3 (GAP-HAMMING problem) *Alice holds a binary string x of n bits in length, and Bob holds a binary string y of n bits in length. The Hamming distance, $H(x, y)$ is defined as $|\{i : x_i \neq y_i\}|$. We are promised that either $H(x, y) \leq n/2 - \sqrt{n}$ or $H(x, y) \geq n/2 + \sqrt{n}$. The goal is for the players to determine which case their input falls in.*

Note that we could solve this problem using summary techniques we have seen already. If we treat the binary strings x and y as vectors, then the vector $(x - y)$ has squared Euclidean norm equal to $H(x, y)$. That is, $H(x, y) = \|x - y\|_2^2$. We can obtain an estimate of the Euclidean

norm (and hence the squared Euclidean norm) with relative error ϵ using space $O(1/\epsilon^2)$. We can answer the GAP-HAMMING problem if this relative error is small enough to distinguish the two cases. This requires $\epsilon \leq \sqrt{n}/H(x,y) \propto 1/\sqrt{n}$. That is, we can solve this problem using a method like AMS Sketch with ϵ chosen so that the space is $O(n)$. However, this is no significant improvement on the trivial approach which simply has Alice send x to Bob, using $O(n)$ bits of communication.

Indeed, this is the best that can be done for this problem: GAP-HAMMING requires $\Omega(n)$ communication from Alice to Bob in the one-round communication complexity model. This result was shown by Woodruff [226], with a simplified proof due to Jayram, Kumar and Sivakumar [138], who reduced the problem to an instance of INDEX.

Lower bound for count distinct and frequency moments. The GAP-HAMMING problem allows a stronger lower bound to be shown for Count Distinct than the direct reduction from INDEX. Given an instance of GAP-HAMMING, Alice and Bob both code up their inputs in the natural way. Alice creates a set as follows: if $x_i = 0$, she inserts an item corresponding to the tuple $(i, 0)$, otherwise she inserts a tuple $(i, 1)$. Bob similarly encodes his set. We can observe that this encoding generates $2H(x,y)$ tuples that occur once over the full input, and a further $n - H(x,y)$ that occur twice. In total, there are $n+H(x,y)$ distinct tuples generated. In one case, the total number of distinct elements is at least $3n/2 + \sqrt(n)$, and in the other it is at most $3n/2 - \sqrt{n}$. Thus, choosing ϵ for an approximate distinct counter to be $\propto 1/\sqrt{n}$ would be sufficient to distinguish these two cases. This implies a space requirement of $\Omega(1/\epsilon^2)$ for the count distinct problem, and hence KMV and HLL are optimal in their dependence on ϵ.

We note that a similar construction and analysis also shows the same $\Omega(1/\epsilon^2)$ space requirement to estimate $\|v\|_p$ vector norms for constant values of p. The hardness for estimating $\|v\|_2$ in turn shows the corresponding hardness for estimating inner product, due to the close connection of these problems: Euclidean norm ($\|v\|_2$) is a special case of inner product since $v \cdot v = \|v\|_2^2$. Hence, inner product $u \cdot v$ is shown to require $\Omega(1/\epsilon^2)$ space in order to estimate with additive error proportional to $\epsilon\|u\|_2\|v\|_2$.

10.5 Augmented Index and Matrix Multiplication

The problem of AUGMENTEDINDEX is used to prove stronger bounds for summaries which allow items to be deleted, also referred to as 'negative updates'.

Definition 10.4 (AUGMENTEDINDEX problem) *Alice holds a binary string x of n bits in length, while Bob holds a string y of $0 \leq i < n$ bits in length, with the promise that $x_j = y_j$ for all $j \leq i$. The goal is for the players to follow a protocol to output $x[i+1]$, the $i+1$st bit from the string x.*

We can view this as equivalent to INDEX with the additional help that Bob knows the prefix of x up to (but not including) the vital bit of interest. As with the basic INDEX problem, the hardness derives from the fact that Alice does not know anything about the critical index $i+1$, and Bob cannot communicate it to her. It turns out that the extra information available to Bob does not help: the communication complexity of this problem remains $\Omega(n)$. However, the extra information does allow us to encode slightly more complex instances of other problems, and hence show stronger lower bounds on summaries that allow deletions or negative updates.

Lower bound for Matrix multiplication. We now describe a lower bound for the matrix multiplication using a reduction to AUGMENTED INDEX, based on an approach of Clarkson and Woodruff [51]. Consider an instance of the Matrix multiplication (Section 6.3). We are given matrices A and B, both of n rows and c columns, specified with incremental updates to their entries. We suppose that we have a summary to estimate the product $A^T B$ with error at most $\epsilon \|A\|_F \|B\|_F$, where $\|\cdot\|_F$ denotes the Frobenius (entrywise) norm, $\|A\|_F = \sqrt{(\sum_{i,j} A_{i,j}^2)}$.

We use the supposed summary to allows us to solve an instance of AUGMENTEDINDEX. Alice will encode her string x into the matrix A by filling it with entries in some canonical fashion, say top to bottom and right to left. The sign of an entry is positive if the corresponding bit in x is 1, and negative if it is 0. The magnitude of the entries are increased from left to right. We set the parameter $r = \log(cn)/(8\epsilon^2)$. In the first $r/\log(cn)$ columns, the magnitude of

all entries is 1; the next $r/\log(cn)$ columns, it is 2; and in the k'th group of $r/\log(cn)$ columns, the magnitude of all entries is 2^k. The remaining $n - r$ columns in A are set to all zeros.

On receiving the summary from Alice, Bob uses his knowledge of the prefix of x to 'remove' all the information in the matrix A corresponding to this prefix. That is, Bob subtracts the quantity in each entry of A that Alice added which corresponds to a bit value that he knows about, since Alice's transformation is completely deterministic. Recall that Bob wants to retrieve the bit corresponding to $x[i+1]$. Let (i^*, j^*) denote the coordinates in matrix A which correspond to $x[i+1]$. Bob then instantiates a matrix B which is entirely 0, except for a single entry of weight 1 in row j^*. Observe that the product $A^T B$ essentially 'reads off' the j^*th column of A^T, which includes the target element.

The permitted error in the reconstruction of $A^T B$ is $\epsilon \|A\|_F \|B\|_F$. By construction, $\|B\|_F = 1$. Meanwhile, let k be such that the largest entry in A is 2^k (after Bob has removed the larger entries). Then $\|A\|_F^2 \leq (cr/\log(cn)) \cdot (1+4+\ldots+2^{2k})$, the size of each 'block' of A, times the squared weights for each block. Then $\|A\|_F^2 \leq \frac{4}{3} cr 2^{2k}/\log(cn)$. Based on the choice of $r = \log(cn)/(8\epsilon^2)$, the permitted squared error is $\epsilon^2 \|A\|_F^2 \|B\|_F^2 \leq c4^k/6$.

We now consider the reconstructed column of A^T found via $A^T B$, which has c entries. We say that an entry is 'bad' if its sign is incorrect in the reconstruction compared to in the original. Observe that since each entry has squared magnitude 4^k, an error in sign contributed at least 4^k to the total squared error. The bound on error from above means that at most a one-sixth fraction of entries can be bad without violating the promise. We can assume that the index of interest is chosen independently of other choices, so the probability that the correct sign is found, and hence the correct value of the bit of interest, is at least 5/6.

This is sufficient to ensure that the instance of AUGMENTEDINDEX is solved with sufficient probability. The size of the instance encoded is the number of non-zero entries in A, which is $O(rc = (c/\epsilon^2)\log(cn))$. This means that any summary for approximate Matrix Multiplication on matrices of size $r \times c$ must use space $\Omega(c\epsilon^{-2}\log(cn))$.

The impact of using AUGMENTEDINDEX is that it allows us to introduce the additional factor of $\log(cn)$ into the hardness bound. If we were to try a similar reduction with an instance of INDEX,

the bound would emerge as $\Omega(c\epsilon^{-2})$. The reduction would involve a matrix A with fewer rows, and all entries with magnitude 1.