How Hard is it to Find (Honest) Witnesses?

Isaac Goldstein[*1], Tsvi Kopelowitz[†2], Moshe Lewenstein[‡3], and Ely Porat[4]

1 **Bar-Ilan University, Ramat Gan, Israel**
`goldshi@cs.biu.ac.il`

2 **University of Michigan, Ann Arbor, USA**
`kopelot@gmail.com`

3 **Bar-Ilan University, Ramat Gan, Israel**
`moshe@cs.biu.ac.il`

4 **Bar-Ilan University, Ramat Gan, Israel**
`porately@cs.biu.ac.il`

Abstract

In recent years much effort has been put into developing polynomial-time conditional lower bounds for algorithms and data structures in both static and dynamic settings. Along these lines we introduce a framework for proving conditional lower bounds based on the well-known 3SUM conjecture. Our framework creates a *compact representation* of an instance of the 3SUM problem using hashing and domain specific encoding. This compact representation admits false solutions to the original 3SUM problem instance which we reveal and eliminate until we find a true solution. In other words, from all *witnesses* (candidate solutions) we figure out if an *honest* one (a true solution) exists. This enumeration of witnesses is used to prove conditional lower bounds on *reporting* problems that generate all witnesses. In turn, these reporting problems are then reduced to various decision problems using special search data structures which are able to enumerate the witnesses while only using solutions to decision variants. Hence, 3SUM-hardness of the decision problems is deduced.

We utilize this framework to show conditional lower bounds for several variants of convolutions, matrix multiplication and string problems. Our framework uses a strong connection between all of these problems and the ability to find *witnesses*.

Specifically, we prove conditional lower bounds for computing partial outputs of convolutions and matrix multiplication for sparse inputs. These problems are inspired by the open question raised by Muthukrishnan 20 years ago [22]. The lower bounds we show rule out the possibility (unless the 3SUM conjecture is false) that almost linear time solutions to sparse input-output convolutions or matrix multiplications exist. This is in contrast to standard convolutions and matrix multiplications that have, or assumed to have, almost linear solutions.

Moreover, we improve upon the conditional lower bounds of Amir et al. [5] for histogram indexing, a problem that has been of much interest recently. The conditional lower bounds we show apply for both reporting and decision variants. For the well-studied decision variant, we show a full tradeoff between preprocessing and query time for every alphabet size > 2. At an extreme, this implies that no solution to this problem exists with subquadratic preprocessing time and $\tilde{O}(1)$ query time for every alphabet size > 2, unless the 3SUM conjecture is false. This is in contrast to a recent result by Chan and Lewenstein [9] for a binary alphabet.

While these specific applications are used to demonstrate the techniques of our framework, we believe that this novel framework is useful for many other problems as well.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

* This research is supported by the Adams Foundation of the Israel Academy of Sciences and Humanities.
† This research is supported by NSF grants CCF-1217338, CNS-1318294, and CCF-1514383.
‡ This research is supported by a BSF grant 2010437 and a GIF grant 1147/2011.

© Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat;
licensed under Creative Commons License CC-BY
24th Annual European Symposium on Algorithms (ESA 2016).
Editors: Piotr Sankowski and Christos Zaroliagis; Article No. 45; pp. 45:1–45:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases 3SUM, convolutions, matrix multiplication, histogram indexing

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.45

1 Introduction

In recent years much effort has been invested towards developing polynomial time lower bounds for algorithms and data structures in both static and dynamic settings. This effort is directed towards obtaining a better understanding of the complexity class P for well-studied problems which seem hard in the polynomial sense. The seminal paper by Gajentaan and Overmars [13] set the stage for this approach by proving lower bounds for many problems in computational geometry conditioned on the 3SUM conjecture. In the 3SUM problem we are given a set A of n integers and we need to establish if there are $a, b, c \in A$ such that $a+b+c = 0$. This problem has a simple $O(n^2)$ algorithm (and some poly-logarithmic improvements in [6, 17]) but no truly subquadratic algorithm is known, where truly subquadratic means $O(n^{2-\epsilon})$ for some $\epsilon > 0$. The 3SUM conjecture states that no truly subquadratic algorithm exists for the 3SUM problem.

Based on this conjecture, there has been a recent extensive line of work establishing conditional lower bounds (CLBs) for many problems in a variety of fields other than computational geometry, including many interesting dynamic problems, see e.g. [1, 2, 3, 4, 19, 23].

1.1 Decision and Reporting Problems

Algorithmic problems come in many flavors. The classic one is the *decision* variant. In this variant, we are given an instance of a problem and we are required to decide if it has some property or not. Some examples include: (1) given a 3-CNF formula we may be interested in deciding if it satisfiable by some truth assignment; (2) given a bipartite graph we may be interested in deciding if the graph has a perfect matching; (3) given a text T and a pattern P we may be interested in deciding if P occurs in T. It is well-known that the first example is NP-complete while the two others are in P. An instance that has the property in question has at least one *witness* that proves the existence of the property. In the examples above a witness is: (1) a satisfying assignment; (2) a perfect matching in the graph; (3) a position of an occurrence of P in T. Sometimes, we are not only interested in understanding if a witness *exists*, but rather we wish to *enumerate* all of the witnesses. This is the *reporting* variant of the problem. In the examples mentioned above the goal of the reporting variant is to: (1) enumerate all satisfying assignments; (2) enumerate all perfect matchings; (3) enumerate all occurrences of P in T. For the first two examples it is known from complexity theory that it is most likely hard to count the number of witnesses (not to mention reporting them) (these are #P-complete problems), while the third example can be solved by classic linear time algorithms.

In this paper we investigate the interplay between the decision and reporting variants of algorithmic problems and present a systematic framework that is used for proving CLBs for these variants. We expect this framework to be useful for proving CLBs on other problems not considered here.

1.2 Our Framework

We introduce and follow a framework that shows 3SUM-hardness of decision problems via their reporting versions. The high-level idea is to reduce an instance of 3SUM to an instance of a reporting problem, and then reduce the instance of a reporting problem to several instances of its decision version using a sophisticated search structure. The outline of this framework is described next.

- **Compact Representation**. One of the difficulties in proving CLBs based on the 3SUM conjecture is that the input universe for 3SUM could be too large for accommodating a reduction to a certain problem. To tackle this, we *embed* the universe using special hashing techniques. This is sometimes coupled with a secondary problem-specific encoding scheme in order to match the problem at hand.
- **Reporting**. The embedding in the first step may introduce false-positives. To tackle this, we report *all* the candidate solutions (witnesses) for the embedded 3SUM instance, in order to verify if a true solution (an honest witness) to 3SUM really exists. This is where we are able to say something about the difficulty of solving reporting problems. This is done by reducing the embedded 3SUM instance to an instance of such a reporting problem, if it provides an efficient way to find all the false-positives. In some cases, such reductions reveal tradeoff relationships between the preprocessing time and reporting/query time.
- **Reporting via Decision**. In this step the goal is to establish 3SUM-hardness of a decision problem. To do so we reduce an instance of the reporting version of the problem to instances of the decision version by creating a data structure on top of the many instances of the decision version. This data structure allows us to efficiently report all of the elements in the output of the instance of the reporting version. By constructing the data structure in different ways we obtain varying CLBs for the decision variants depending on the specific structure that we use.

By following this route we introduce new CLBs for some important problems which are discussed in detail in Section 2. We point out that the embedding in the first step follows along the lines of [23] and [19]. However, in some cases we also add an additional encoding scheme to fit the needs of the specific problem at hand.

Implications. In Section 2 we discuss three applications from two different domains which utilize our framework for proving CLBs, thereby demonstrating the usefulness of our framework. Table 1 summarizes these results. Of particular interest are new results on Histogram Indexing (defined in Section 2) which, together with the algorithm of [9], demonstrate a sharp separation when allowing truly subquadratic preprocessing time between binary and trinary alphabet settings. Moreover, our framework is the first to obtain a CLB for the reporting version, which, as opposed to the decision variant, also holds for the binary alphabet case.

2 Applications

Convolution Problems

The *convolution* of two vectors $u, v \in \{\mathbb{R}^+ \cup \{0\}\}^n$ is a vector w, such that $w[k] = \sum_{i=0}^{k} u[i]v[k-i]$ for $0 \leq k \leq 2n-2$. Computing the convolution of u and v takes $O(n \log n)$ time using the celebrated FFT algorithm. Convolutions are used extensively in many areas including signal processing, communications, image compression, pattern matching, etc. A *convolution witness* for the kth entry in w is a pair (a, b) such that $a+b = k$ and $u[a] \cdot v[b] > 0$. In other words, the witnesses of entry k in w are all values i that contribute a non-zero value to the summation $w[k] = \sum_{i=0}^{k} u[i]v[k-i]$. The first convolution problem we consider is the *convolution witnesses problem* which is defined as follows.

▶ **Definition 1.** In the **convolution witnesses problem** we preprocess two vectors $u, v \in \{\mathbb{R}^+ \cup \{0\}\}^n$ and their convolution vector w, so that given a query integer $0 \leq k \leq 2n-2$, we list *all* convolution witnesses of index k in w.

We prove the following CLB for the convolution witnesses problem that holds even if u and v are binary vectors and all numbers in w are non-negative integers.

▶ **Theorem 2.** *Assume the 3SUM conjecture is true. Then for any constant $0 < \alpha < 1$, there is no algorithm solving the convolution witnesses problem with $O(n^{2-\alpha})$ expected preprocessing time and $O(n^{\alpha/2-\Omega(1)})$ expected amortized query time* ***per witness****.*

Theorem 2 implies that when using only truly subquadratic preprocessing time one is required to spend a significant polynomial amount of time on every single witness. In particular, this means that, assuming the 3SUM conjecture, one cannot expect to find witnesses much faster than following the naive algorithm for computing convolution naïvely according to the convolution definition. This is in contrast to the decision version of the problem, where we only ask if a witness exists. This variant is easily solved using constant query time after a near linear time preprocessing procedure (computing the convolution itself).

Another variation of the convolution problem which we consider is the *sparse convolution problem*. There are two different problems named sparse convolution, both appearing as open questions in a paper by Muthukrishnan [22]. In the first, which is now well understood, we are given Boolean vectors u and v of lengths N and M, where $M < N$. There are n ones in u, m ones in v and z ones in w, where w is the Boolean convolution vector of u and v. The goal is to report the non-zero elements in w in $\tilde{O}(z)$ time. This problem has been extensively studied, and the goal has been achieved; see for example [9, 11, 15]. The second variant which we call *partial convolutions* is as follows.

▶ **Definition 3.** The **partial convolution problem** on two vectors u and v of real numbers (of length N and M respectively, where $M < N$) and a set S of indices is to compute, for each $i \in S$, the value of the i-th element in the convolution of u and v.

Muthukrishnan in [22] asked if it is possible to compute a partial convolution significantly faster than the time needed to compute a (classic) convolution. We prove a CLB based on the 3SUM conjecture, that holds also for the special case of Boolean vectors, and, therefore, also for the special case in which we only want to know if the output values at indices in S are zero or more. Moreover, we focus on the important variant of this problem that deals with the case where the two input vectors have only $n = O(N^{1-\Omega(1)})$ ones and are both given implicitly (specifying only the indices of the ones). Our results also extend to the indexing version of the partial convolution problem, which we call the *partial convolution indexing problem*, and is defined as follows.

▶ **Definition 4.** The partial convolution indexing problem is to preprocess an N-length vector u of real numbers and a set of indices S to support the following queries: given an M-length vector v ($M < N$) of real numbers, for each $i \in S$ compute the value of the i-th element of the convolution of u and v.

Once again this variant already relevant when the input is Boolean and sparse, i.e. u and v have $n = O(N^{1-\Omega(1)})$ ones and are represented implicitly by specifying their indices.

We prove the following CLBs for these problems with the help of our framework.

▶ **Theorem 5.** *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial convolution problem with $O(N^{1-\Omega(1)})$ time, even if $|S|$ and the number of ones in both input vectors are less than $N^{1-\Omega(1)}$.*

▶ **Theorem 6.** *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial convolution indexing problem with $O(N^{2-\Omega(1)})$ preprocessing time and $O(N^{1-\Omega(1)})$ query time, even if both $|S|$ and the number of ones of the input vectors are $O(N^{1-\Omega(1)})$.*

As mentioned above, the convolution of vectors of length N can be computed in $\tilde{O}(N)$ time with the FFT algorithm. However, in the partial convolution problem and partial convolution indexing problem, despite the input vectors being sparse and represented sparsely (specifying only the $O(N^{1-\Omega(1)})$ indices of the ones in each vector), and despite the portion of the output we need to compute being sparse ($|S| = O(N^{1-\Omega(1)})$), no linear time algorithm (in $n = O(N^{1-\Omega(1)})$) exists, unless the 3SUM conjecture is false.

Notice that the partial convolution problem and its indexing variant are *decision* problems, since they require a *decision* for each location $i \in S$, whether $w[i] > 0$ or not. This is in contrast to the convolution witnesses problem, which is a reporting problem, as it requires the reporting of *all* of the witnesses for $w[i]$.

To prove CLBs for the convolution problems we follow our framework. That is, we first use a hash function to embed a 3SUM instance to a *smaller* universe. This mapping introduces false-positives, which we enumerate by utilizing the reporting problem of convolution witnesses. To solve the reporting version we reduce it to several instances of a decision problem, partial convolution or its indexing variant, by constructing a suitable data structure. Tying it all together leads to CLBs for both the reporting and decision problems.

Matrix Problems

We also present some similar CLBs for matrices.

▶ **Definition 7.** The **partial matrix multiplication problem** on two $N \times N$ matrices A and B of real numbers and a set of entries $S \subseteq N \times N$ is to compute, for each $(i, j) \in S$, the value $(A \times B)[i, j]$.

The indexing variant of this problem is defined as follows.

▶ **Definition 8.** The **partial matrix multiplication indexing problem** is to preprocess an $N \times N$ matrix A of real numbers and a collection $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ of sets of entries, where $S_i \subseteq N \times N$, so that given a sequence $B_1, \ldots, B_k$ of $N \times N$ matrices of real numbers, we enumerate the entries of $A \times B_i$ that correspond to S_i.

For $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ let $SIZE(S) = \sum_{i=1}^{k} |S_i|$. We prove the following CLBs, which hold also for the special case of Boolean multiplication assuming that the input is given implicitly by specifying only the indices of the ones.

▶ **Theorem 9.** *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial matrix multiplication problem running in $O(N^{2-\Omega(1)})$ expected time, even if $|S|$ and the number of ones in the input matrices is $O(N^{2-\Omega(1)})$.*

▶ **Theorem 10.** *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial matrix multiplication indexing problem with $O(SIZE(S))$ preprocessing time and $O(N^{2-\Omega(1)})$ query time.*

Matrix multiplication, and in particular Boolean matrix multiplication, can be solved in $\tilde{O}(n^\omega)$ time, where $\omega \approx 2.373$ [14, 25]. Many researchers believe that the true value of ω is 2. This belief implies that the running time for computing the product of two Boolean matrices is proportional to the size of the input matrices and the resulting output. However, our results demonstrate that such a result is unlikely to exist for sparse versions of the problem, where the number of ones in the matrices is $O(N^{2-\Omega(1)})$ and we are interested in only a partial output matrix (only $O(N^{2-\Omega(1)})$ entries of the matrix product).

To prove Theorem 9 and 10 we follow our framework. The process is very similar to the path for proving CLBs for convolution problems. In fact, instead of considering a reporting version of the partial matrix multiplication problem for proving these CLBs, we once again utilize the reporting problem of convolution witnesses. However, this time we transform the convolution witnesses to the matrix multiplication problems using a more elaborate data structure. The main difficulty in this transformation is to guarantee the sparsity of both the input and the required output. This transformation illustrates how a reporting version of a problem can be used to prove CLBs for decision versions of other problems, by changing the way we look for honest witnesses.

String Problems

Another application of our framework, which is seemingly unrelated to the previous two, is the problem of histogram indexing. A *histogram*, also called a *Parikh vector*, of a string T over alphabet Σ is a $|\Sigma|$-length vector containing the character count of T. For example, for $T = abbbacab$ the histogram is $\psi(T) = (3, 4, 1)$.

► **Definition 11.** In the **histogram indexing problem** we preprocess a string T to support the following queries: given a query Parikh vector ψ, return whether there is a substring T' of T such that $\psi(T') = \psi$.

► **Definition 12.** In the **histogram indexing reporting problem** we preprocess a string T to support the following queries: given a query Parikh vector ψ, report indices of T at which a substring T' of T begins such that $\psi(T') = \psi$.

The problem of histogram indexing (not the reporting version) is sometimes called *jumbled indexing*. It has received much attention in recent years. For example, for binary alphabets – that is histograms of length 2 – there is a straightforward algorithm with $O(n^2)$ preprocessing time and constant query time, see [10]. Burcsi et al. [8] and Moosa and Rahman [20] improved the preprocessing time to $O(n^2/\log n)$. Using the four-Russian trick a further improvement was achieved by Moosa and Rahman [21]. Then, using a connection to the recent improvement of all-pairs-shortest path by Williams [24], as observed by Bremner et al. [7] and by Hermelin et al. [16], the preprocessing time was further reduced to $O(\frac{n^2}{2^{\Omega(\log n)^{0.5}}})$. Finally, Chan and Lewenstein [9] presented an $O(n^{1.859})$ preprocessing time algorithm for the problem with constant query time. For non-binary alphabets some progress was achieved in the work by Kociumaka et al. [18] and even further achievement was shown in [9]. On the negative side, some CLBs were recently shown by Amir et al. [5].

We follow our framework and first obtain CLBs for the reporting version of histogram indexing. This is the first time CLBs are shown for the reporting version. Moreover, these CLBs apply to binary alphabets, as opposed to the decision version in which there currently is no CLB known for binary alphabets. The CLBs for the reporting version admit a full tradeoff between preprocessing and query time. For the decision variant, we improve upon the CLB by Amir et al. [5] by presenting full-tradeoffs between preprocessing and query

Table 1 Summary of CLBs proved in this paper. In this table N is the size of vectors, strings and the dimension of matrices. #1 refers to the number of ones in the input. The rows in this table are interpreted to mean that there is no data structure that beats these preprocessing, query, and reporting (if exists) complexities at the same time. For partial convolution and matrix multiplication the CLB on the preprocessing time should be interpreted as a CLB on the total running time as these are offline problems.

Problem *(Type)*	Preprocessing Time	Query Time	Reporting Time	Remarks
Conv. Witnesses *(Reporting)*	$\Omega(N^{2-\alpha})$	$\Omega(N^{1-\alpha/2})$	$\Omega(N^{\alpha/2-o(1)})$	[Theorem 2] $0 < \alpha < 1$
Partial Conv. *(Decision)*	$\Omega(N^{1-o(1)})$	—	—	[Theorem 5] Sparse input: $\#1 < N^{1-\Omega(1)}$; Sparse required output: $\|S\| < N^{1-\Omega(1)}$
Partial Conv. Indexing *(Decision)*	$\Omega(N^{2-o(1)})$	$\Omega(N^{1-o(1)})$	—	[Theorem 6] Sparse input: $\#1 < N^{1-\Omega(1)}$; Sparse required output: $\|S\| < N^{1-\Omega(1)}$
Partial Matrix Mult. *(Decision)*	$\Omega(N^{2-o(1)})$	—	—	[Theorem 9] Sparse input: $\#1 < N^{2-\Omega(1)}$; Sparse required output: $\|S\| < N^{2-\Omega(1)}$
Partial Matrix Mult. Indexing *(Decision)*	$\Omega(SIZE(S))$	$\Omega(N^{2-o(1)})$	—	[Theorem 10] Sparse input: $\#1 < N^{2-\Omega(1)}$; Sparse required output: $\|S_i\| < N^{2-\Omega(1)}$; $SIZE(S) = \sum_{i=1}^{k} \|S_i\|$
Histogram Reporting *(Reporting)*	$\Omega(N^{2-\frac{2\gamma}{\ell+\gamma}-o(1)})$	$\Omega(N^{1-\frac{\gamma}{\ell+\gamma}-o(1)})$	$\Omega(N^{\frac{\gamma\ell}{\ell+\gamma}-\frac{2\gamma}{\ell+\gamma}-o(1)})$	[Theorem 13] alphabet size: $\ell \geq 2$; $0 < \gamma < \ell$
Histogram Indexing *(Decision)*	$\Omega(N^{2-\frac{2(1-\alpha)}{\ell-1-\alpha}-o(1)})$	$\Omega(N^{1-\frac{1+\alpha(\ell-3)}{\ell-1-\alpha}-o(1)})$	—	[Theorem 14] alphabet size: $\ell > 2$; $0 \leq \alpha \leq 1$

time based on the standard 3SUM conjecture. Specifically, our new CLB implies that no solution to the histogram indexing problem exists with subquadratic preprocessing time and $\tilde{O}(1)$ query time for every alphabet size bigger than 2, unless the 3SUM conjecture is false. This demonstrates a sharp separation between binary and trinary alphabets, since Chan and Lewenstein [9] introduced an algorithm for histogram indexing on binary alphabets with $\tilde{O}(n^{1.859})$ preprocessing time and constant query time.

The CLBs are summarized by the following theorems.

▶ **Theorem 13.** *Assume the 3SUM conjecture is true. Then the histogram reporting problem for an N-length string and constant alphabet size $\ell \geq 2$ cannot be solved using $O(N^{2-\frac{2\gamma}{\ell+\gamma}-\Omega(1)})$ preprocessing time, $O(N^{1-\frac{\gamma}{\ell+\gamma}-\Omega(1)})$ query time and $O(N^{\frac{\gamma\ell}{\ell+\gamma}-\frac{2\gamma}{\ell+\gamma}-\Omega(1)})$ reporting time per item, for any $0<\gamma<\ell$.*

▶ **Theorem 14.** *Assume the 3SUM conjecture holds. Then the histogram indexing problem for a string of length N and constant alphabet size $\ell \geq 3$ cannot be solved with $O(N^{2-\frac{2(1-\alpha)}{\ell-1-\alpha}-\Omega(1)})$ preprocessing time and $O(N^{1-\frac{1+\alpha(\ell-3)}{\ell-1-\alpha}-\Omega(1)})$ query time.*

The main structure of these proofs follows our framework. We first embed a 3SUM instance and encode it in a string with limited length. We then report the false-positives using the reporting variant of the histogram indexing problem, which implies CLBs for this variant. Finally, we reduce the reporting version to the decision version thereby obtaining CLBs for the decision version. The reduction utilizes a sophisticated data structure for reporting witnesses using many instances of the decision version.

3 Preliminaries

In the basic 3SUM problem we are given a set A of n integers and we need to answer whether there are $a, b, c \in A$ such that $a+b+c=0$. In a common variant of the classic problem, which we also denote by 3SUM, three arrays A, B and C are given and we need to answer whether there are $a \in A, b \in B, c \in C$ such that $a+b+c=0$. Both versions have the same computational cost (see [13]). There are some other variants of the 3SUM problem shown to be as hard as 3SUM up to poly-logarithmic factors. One such variant is Convolution3SUM, shown to be hard by Pătraşcu [23], see also [19]. In Convolution3SUM A is an ordered set and we need to answer whether there exist indices $0 \leq i, j \leq n-1$ such that $A[i]+A[j]=A[i+j]$. We also define DiffConv3SUM, in which we are given an ordered set A and we need to verify whether there exists $0 \leq i, k \leq n-1$ such that $A[k]-A[i]=A[k-i]$. It is easy to see that this is equivalent to Convolution3SUM.

Let $\mathcal{H}$ be a family of hash functions from $[u] \to [m]$.

$\mathcal{H}$ is called *linear* if for any $h \in \mathcal{H}$ and any $x, x' \in [u]$, we have $h(x)+h(x') \equiv h(x+x') \pmod{m}$. $\mathcal{H}$ is called *almost-linear* if for any $h \in \mathcal{H}$ and any $x, x' \in [u]$, we have either $h(x)+h(x') \equiv h(x+x')+c_h \pmod{m}$, or $h(x)+h(x') \equiv h(x+x')+c_h+1 \pmod{m}$, where c_h is an integer that depends only on the choice of h. For a function $h : [u] \to [m]$ and a set $S \subset [u]$ where $|S|=n$, we say that $i \in [m]$ is an overflowed value of h if $|\{x \in S : h(x)=i\}| > 3n/m$. $\mathcal{H}$ is called *almost-balanced* if for a random $h \in \mathcal{H}$ and any set $S \subset [u]$ where $|S|=n$, the expected number of elements from S that are mapped to overflowed values is $O(m)$. See [19] for constructions of families that are almost-linear and almost-balanced (see also [6, 12]).

For simplicity of presentation, and following the footsteps of previous papers that have used such families of functions [6, 23], we assume for the rest of the paper that almost linearity implies that for any $h \in \mathcal{H}$ and any $x, x' \in [u]$ we have $h(x)+h(x') \equiv h(x+x') \pmod{m}$.

There are actually two assumptions taking place here. The first is that there is only one option of so-called linearity. Overcoming this assumption imposes only a constant factor overhead. The second assumption is that $c_h = 0$. However, the constant c_h only affects offsets in our algorithm in a straightforward and not meaningful way, so we drop it in order to avoid clutter in our presentation.

4 Convolution Witnesses

We first prove a CLB for the convolution witnesses problem. We begin with a lemma which has elements from the proof of Pătraşcu's reduction [23] and from [6]. However, the lemma diverges from [23] by treating the hashed subsets differently. Specifically, many special 3SUM subproblems are created and then reduced to convolution witnesses.

We say that a binary vector of length n is *r-sparse* if it contains at most r 1's. An instance of convolution witnesses problem (u, v, w) is *(n, R)-sparse* if u and v are both of length n and n/R-sparse.

▶ **Lemma 15.** *Let sequence $A = \langle x_1, \cdots, x_n \rangle$ be an instance of Convolution3SUM. Let $R = O(n^\delta)$, where $0 < \delta < 0.5$ is a constant. There exists a truly subquadratic reduction from the instance A to $O(R^2)$ (n, R)-sparse instances of convolution witnesses problem for which we need to report $O(n^2/R)$ witnesses (over all instances).*

Proof. We use an almost-linear, almost-balanced, hash function $h : U \to [R]$ and create R buckets $B_0, \cdots, B_{R-1}$ where each B_a contains the indices of all elements $x_i \in A$ for which $h(x_i) = a$. Since h is almost-balanced the expected overall number of elements in buckets with more than $3n/R$ elements is $O(R)$. For each index i in an overflowed bucket, we verify whether $x_i + x_j = x_{i+j}$ for every other j in $O(n)$ time. Hence, we verify whether any index in an overflowed bucket is part of a Convolution3SUM solution in $O(nR)$ expected time. Since $R = O(n^{1-\Omega(1)})$ the expected time is truly subquadratic time.

We now assume that every bucket contains at most $3n/R$ elements. From the properties of almost-linear hashing, if $x_i + x_j = x_{i+j}$ then $h(x_i) + h(x_j) \bmod R = h(x_{i+j}) \bmod R$. Hence, if $x_i + x_j = x_{i+j}$ then $i \in B_a, j \in B_b$ implies that $i + j \in B_{a+b \bmod R}$.

Every three buckets form an instance of 3SUM and are uniquely defined by a and b. Hence, there are $R(R-1)/2 = O(R^2)$ 3SUM subproblems each on $O(n/R)$ elements from the small universe $[n]$. However, h may generate false positives. So, we must be able to verify that any 3SUM solution (a witness) for any instance is indeed a solution (an honest witness) for the problem on A. The number of false positives is expected to be $O(n^2/R)$ over all $O(R^2)$ instances, see [6]. So, we need an efficient tool to report each such witness in order to be able to solve Convolution3SUM.

To obtain such a tool, we reduce the problem to the convolution setting in the following way. We generate a characteristic vector v_a of length n for every set B_a ($v_a[i] = 1$ if $i \in B_a$ and $v_a[i] = 0$ otherwise, for $0 \le i < n$). This vector will be $3n/R$-sparse, since $|B_a| \le 3n/R$. Note that: $i \in B_a, j \in B_b$ and $\mathrm{i+j \in B_{a+b \bmod R}} \iff \mathrm{v_a[i] = 1, v_b[j] = 1}$ and $\mathrm{v_{a+b \bmod R}[i+j] = 1}$.

Now, for each pair of vectors, v_a and v_b, we generate their convolution. Let $v = v_a * v_b$ be the convolution of v_a and v_b, and let $\ell = v[i+j]$. If $v_{a+b \bmod R}[i+j] = 1$, then we need to extract the ℓ witnesses of $v[i+j]$. For each witness (i, j) we check whether $x_i + x_j = x_{i+j}$. We note that if, while verifying, we discover that the overall number of the false-positives exceeds expectation (cn^2/R, for some constant c) by more than twice we rehash.

Thus, we see that Convolution3SUM can be solved by generating $O(R^2)$ (n, R)-sparse instances of convolution witnesses problem. These instances are computed in $O(nR^2)$ time, which is truly subquadratic as $R = O(n^\delta)$ for $\delta < 1/2$. ◀

It now follows that:

▶ **Theorem 2** (restated). *Assume the 3SUM conjecture is true. Then for any constant $0 < \alpha < 1$, there is no algorithm solving the convolution witnesses problem with $O(n^{2-\alpha})$ expected preprocessing time and $O(n^{\alpha/2-\Omega(1)})$ expected amortized query time **per witness**.*

Proof. We make use of Lemma 15 and its parameter R. In particular, the total cost of solving Convolution3SUM is at most $O(R^2 \cdot P(n,R) + n^2/R \cdot Q(n,R))$ expected time, where $P(n,R)$ is the time needed to preprocess an (n,R)-sparse instance of a convolution witness and $Q(n,R)$ is the time per witness query for an (n,R)-sparse instance of a convolution witness.

If we choose $R = n^{\alpha/2-\Omega(1)}$ we have that for $P(n) = O(n^{2-\alpha})$ and $Q(n) = O(n^{\alpha/2-\Omega(1)})$ we solve Convolution3SUM in $O(n^{2-\Omega(1)})$ time which is truly subquadratic. ◀

5 From Reporting to Decision I: Hardness of Partial Convolutions

We further consider the problem of reporting witnesses for convolutions. However, now we use the third step of our framework. We will construct a search data structure over decision problems which will allow us to efficiently search for witnesses. This will be our method for proving CLBs for the decision problems of partial convolutions [22]. Specifically, we intend to generate a data structure that uses convolutions on small sub-vectors of the input vectors in order to solve the problem. However, the data structure cannot be fully constructed as it will be too large. Hence, the construction is partial and we defer some of the work to the query phase.

We start with Lemma 15, and focus on an (n,R)-sparse instance of the convolution witnesses problem(u,v,w). We generate a specialized search tree for efficiently finding witnesses, which is created in an innovative way exploiting the sparsity of the input.

5.1 Search Tree Construction

Assume, without loss of generality, that n is a power of 2. We construct a binary tree in the following way. First, we generate the root of the tree with the convolution of v and u. Then we split u into 2 sub-vectors, say u_1 and u_2, each containing exactly $n/(2R)$ 1s. For each sub-vector we generate nodes that are children of the root, where the first node contains the convolution of v and u_1 and the second node contains the convolution of v and u_2. We continue this construction recursively so that at the ith recursive level we partition u into 2^i sub-vectors each containing $n/(2^iR)$ 1s. A vertex at level i represents the convolution of v and a sub-vector u_A containing $n/2^iR$ 1s. The vertex has two children, one represents the convolution of v and the sub-vector of u_A with the first $n/2^{i+1}R$ 1s of u_A (denoted by $u_{A,1}$). The other represents the convolution of v and the rest of u_A with the other $n/2^{i+1}R$ 1s (denoted by $u_{A,2}$). We stop the construction at the leaf level in which u is split to sub-vectors that each one of them contains X/R 1s from u, for some $X < n$ to be determined later. Calculating the convolution in each vertex is done bottom-up. First, we calculate the convolution for each vertex in the leaf level. Then, we use these results to calculate the convolution of the next level upwards. Specifically, if we have vertex that represent the convolution v and some sub-vector u_A and it has two children one which represents the convolution of v and $u_{A,1}$ and the other which represents the convolution of v and $u_{A,2}$, then $(v * u_A)[k] = (v * u_{A,1})[k] + (v * u_{A,2})[k - l_1]$ for every $k \in [0, n + l_1 + l_2 - 1]$, where l_1 and l_2 are the lengths of $u_{A,1}$ and $u_{A,2}$ respectively, and we consider the value of out of range entries as zero. This way we continue to calculate all the convolutions in the tree until reaching its root.

Construction Time. It is straightforward to verify that the total cost of the construction procedure is dominated by the time of constructing the lowest level of the binary tree. In this level, we have n/X sub-vectors of u as each of them has X/R 1's and the total number of 1s in u is n/R. We calculate the convolution of v with each of these sub-vectors, which can be done in $\tilde{O}(n)$ time. Thus, the total time needed to build the tree is $\tilde{O}(n^2/X)$. herefore, the total time for calculating the binary trees for all $O(R^2)$ (n, R)-sparse instances of the convolution witnesses problem is $\tilde{O}(R^2n^2/X)$.

Witness Search. To search for a witness we begin from the root of the binary tree and traverse down to a leaf containing a non-zero value in the result of the convolution at the query index (adjusting the index as needed while moving down the structure). The search for a leaf costs logarithmic time per query (as the tree has logarithmic height and in each level we just need to find a child with a non-zero value in the convolution it represents in the specific index of interest). Within the leaf, representing the convolution of v and some sub-vector u_A of u we can simply find a witness in $\tilde{O}(X/R)$ time as u_A contains just X/R 1s. Thus, as we have $O(n^2/R)$ false-positives over all $O(R^2)$ instances, the total time for finding all them is $\tilde{O}(n^2X/R^2)$.

Consequently, using the binary tree for solving Convolution3SUM will cost $\tilde{O}(R^2n^2/X + n^2X/R^2)$ time, which for $X = R^2$ is $\tilde{\Theta}(n^2)$ time. Since the tradeoff between the preprocessing time and query time meets at n^2, any improvement to the running time of either of them will imply a subquadratic solution for the Convolution3SUM problem.

5.2 Conditional Lower Bounds for Partial Convolution

As a consequence of our discussion above we obtain the following results regarding partial convolution and its indexing variant:

▶ **Theorem 5** (restated). *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial convolution problem with $O(N^{1-\Omega(1)})$ time, even if $|S|$ and the number of ones in both input vectors are less than $N^{1-\Omega(1)}$.*

Proof. We make use of Lemma 15. In order to construct the binary tree as described in Section 5.1, we need to be able compute the convolution of v with some sub-vector of u for each leaf in the tree (all other convolution can be calculated efficiently from the convolutions in the leaves as described in the previous section). Recall that both input vectors have length $N = n$, n/R 1s (which is $O(N^{1-\Omega(1)})$ for $R = n^a$, where a is a positive constant), and we are interested in finding their convolution result only at the $O(n/R)$ indices (that is, $|S| = O(N^{1-\Omega(1)})$). If we preprocess the input for partial convolution in truly sublinear time (for example, proportional to n/R) then the total time for constructing all the search trees will be $O(R^2n^{2-\Omega(1)}/X)$ while the total query time will remain $O(n^2X/R^2)$. Choosing $X = n^c$ for small enough constant c and setting $R = X$, we obtain a subquadratic solution to Convolution3SUM. ◀

▶ **Theorem 6** (restated). *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial convolution indexing problem with $O(N^{2-\Omega(1)})$ preprocessing time and $O(N^{1-\Omega(1)})$ query time, even if both $|S|$ and the number of ones of the input vectors are $O(N^{1-\Omega(1)})$.*

Proof. Use Lemma 15 and the previous discussion. If the preprocessing time for the partial convolution indexing problem is truly subquadratic and queries are answered in truly sublinear time then the total time for constructing all the structures for all $O(R^2)$ instances is $O(R^2[n^{2-\Omega(1)} + n^{1-\Omega(1)} \cdot n/X])$ while the total time for all of the queries remains $O(n^2X/R^2)$ (note that $N = n$). Choosing $X = n^c$ for small enough constant c and setting $R = X$, we obtain a subquadratic algorithm for Convolution3SUM. ◀

6 From Reporting to Decision II: Hardness of Partial Matrix Multiplication

We present another transformation from the reporting problem of convolution witnesses to decision problems. This time we prove CLBs for the partial matrix multiplication and its indexing variant. The main difficulty in this transformation is to ensure the sparsity of both input and required output. The CLBs that we prove are stated as follows (full details and proofs will appear in the full version of this paper).

▶ **Theorem 9** (restated). *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial matrix multiplication problem running in $O(N^{2-\Omega(1)})$ expected time, even if $|S|$ and the number of ones in the input matrices is $O(N^{2-\Omega(1)})$.*

▶ **Theorem 10** (restated). *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial matrix multiplication indexing problem with $O(SIZE(S))$ preprocessing time and $O(N^{2-\Omega(1)})$ query time.*

7 Hardness of Data Structures for Histogram Indexing

In order to prove a CLB for both the histogram indexing problem and the histogram (indexing) reporting problem, we will first focus on reducing 3SUM to the histogram reporting problem, and then turn our focus to reducing the the histogram reporting problem to the histogram indexing problem.

7.1 Reducing Convolution3SUM to Histogram Reporting

We are given an ordered set A of integers $x_1, x_2, ..., x_n$ for which we want to solve DiffConv3SUM. Our methodology here is to encode the input integers into a compact string S so that histogram indexing with carefully chosen query patterns implies a solution to DiffConv3SUM. Since the size of the universe of the input integers can be as large as n^3, we hash down the universe size while (almost) maintaining the linearity property of the input. To do this, we make use of an almost-linear almost-balanced hash function $h : U \to [R]$ as defined in Section 3, and apply h to all of the input integers.

After utilizing h to compress the input range, we are ready to encode the input and create the string S. To do this, we encode each $h(x_k)$ separately, and then concatenate the encodings in the same order as their corresponding original integers in A. We use the following encoding scheme, using an alphabet $\Sigma = \{\sigma_0, \sigma_1, , ..., \sigma_{\ell-1}\}$. Some other encoding schemes, which surprisingly provide the same bounds, will be presented in the full version of this paper.

Encoding 1. The encoding will consist of two separate partial encodings concatenated together. The first partial encoding is partitioned into ℓ parts which together will represent $h(x_k)$ in base $R^{1/\ell}$. For $0 \le j \le \ell - 1$ the jth part of this first partial encoding is a unary representation of $p_{j,h(x_k)} = \lfloor h(x_k)/R^{j/\ell} \rfloor \bmod R^{1/\ell}$ using σ_j, and is denoted by $enc(j,h(x_k)) = \sigma_j^{p(j,h(x_k))}$. The first partial encoding of $h(x_k)$, which we also call a *regular encoding* of $h(x_k)$, is $enc_\ell(h(x_k)) = enc(0,h(x_k))enc(1,h(x_k)) \cdots enc(\ell-1,h(x_k)) = \sigma_0^{p_{0,h(x_k)}} \sigma_1^{p_{1,h(x_k)}} \cdots \sigma_{\ell-1}^{p_{\ell-1,h(x_k)}}$.

For the second partial encoding we encode the *complement* of each $enc(j,h(x_k))$ which is the unary representation of $\bar{p}_{j,h(x_k)} = R^{1/\ell} - (\lfloor h(x_k)/R^{j/\ell} \rfloor \bmod R^{1/\ell})$ using σ_j, and is denoted by $\overline{enc}(j,h(x_k))$. The second partial encoding of $h(x_k)$, which we also call a *complement encoding* of $h(x_k)$, is $\overline{enc_\ell}(h(x_k)) = \overline{enc}(0,h(x_k))\overline{enc}(1,h(x_k)) \cdots \overline{enc}(\ell-1,h(x_k)) = \sigma_0^{\bar{p}_{0,h(x_k)}} \sigma_1^{\bar{p}_{1,h(x_k)}} \cdots \sigma_{\ell-1}^{\bar{p}_{\ell-1,h(x_k)}}$.

The full encoding of $h(x_k)$ is the concatenation of $\overline{enc_\ell}(h(x_k))$ and $enc_\ell(h(x_k))$ which we denote by $ENC_\ell(h(x_k))$. Finally, the string S is set to be $ENC_\ell(h(x_1))ENC_\ell(h(x_2)) \cdots ENC_\ell(h(x_n))$. The size of S is clearly $N = O(\ell \cdot R^{\frac{1}{\ell}} n)$. We denote the substring of S starting at the location of the beginning of $enc_\ell(h(x_i))$ and ending at the location of the end of $\overline{enc_\ell}(h(x_j))$ by $S_{i,j}$.

Consider a Parikh vector v_k obtained from x_k and h where the rth element has a count of $\bar{p}_{r,h(x_k)} + R^{1/\ell} \cdot (k-1)$. We say that v_k *represents* x_k. For a vector $w = (w_0, w_1, ..., w_m)$ we define $w^{>>1} = (0, w_0, w_1, ..., w_{m-1})$. We also define the *carry set* of v_k to be $V_k = \{v_k + R^{1/\ell} u - u^{>>1} \mid u = (u_0, u_1, ..., u_{\ell-2}, 0),\ u_i \in \{0,1\}\ 0 \le i < \ell - 1\}$. It is easy to see that $|V_k| = 2^{\ell-1}$ and that V_k can be obtained from v_k in $O(\ell \cdot 2^{\ell-1})$ time. We call v_k the *base* of V_k. We have the following lemma regarding V_k:

▶ **Lemma 16.** *If there exists a pair x_i, x_j such that $x_k = x_j - x_i$ and $k = j - i$, then the Parikh vector of $S_{i,j}$ must be in V_k.*

Proof. Since h is linear we know that $h(x_k) = h(x_j) - h(x_i)$. This is equivalent to saying that $R + R^{\frac{\ell-1}{\ell}} - h(x_k) = R + R^{\frac{\ell-1}{\ell}} - [h(x_j) - h(x_i)] = (R + R^{\frac{\ell-1}{\ell}} - h(x_j)) + h(x_i)$. In $S_{i,j}$ we have the full encoding of all integers $x_{i+1}, ..., x_{j-1}$. There are exactly $k-1$ integers between x_i and x_j. Therefore, each of them adds $R^{1/\ell}$ occurrences of each σ_r $(0 \le r \le l-1)$ to $S_{i,j}$. In addition to the full encodings of these integers we have two more partial encodings: $enc_\ell(h(x_i))$ and $\overline{enc_\ell}(h(x_j))$. Notice that $enc_\ell(h(x_i))$ and $\overline{enc_\ell}(h(x_j))$ represent $h(x_i)$ and $R + R^{\frac{\ell-1}{\ell}} - h(x_j)$, respectively, in base $R^{1/\ell}$. If we look at the vector v_k (the base of V_k) after subtracting $(k-1)R^{1/\ell}$ from the count of each character, we obtain the representation of $R + R^{\frac{\ell-1}{\ell}} - h(x_k)$ in base $R^{1/\ell}$, which intuitively implies that v_k is the Parikh vector that we are looking for. However, it is possible to generate a carry at each of the ℓ digits of the base $R^{1/\ell}$ during the addition of $(R + R^{\frac{\ell-1}{\ell}} - h(x_j)) + h(x_i)$. To handle these carries we consider all possible 2^ℓ carry scenarios and generate a vector for each of the $2^{\ell-1}$ scenarios. These carry scenarios are exactly represented by the vectors in V_k, as each vector u in the definition of V_k specifies the indices in which we have a carry. Hence, the Parikh vector of $S_{i,j}$ must be one of the vectors in V_k. ◀

Thus, we preprocess S with an algorithm for histogram reporting, and then query the resulting data structure with all the vectors in V_k, whose base v_k represents some x_k, in an attempt to decide if x_k is part of a solution to DiffConv3SUM. The reported locations are classified into two types:

Candidates: Locations where the histogram match begins and ends exactly between the complement and regular encodings of two input integers. All these locations correspond to x_i and x_j such that for the particular $h(x_k)$ for which the query was constructed, we have $h(x_k) = h(x_j) - h(x_i)$ and also $k = j - i$.

Encoding Errors: All matches that are not candidates.

While encoding errors clearly do not provide a solution for DiffConv3SUM on A, candidates may also not be suitable for a solution since the function h introduces false-positives. The following lemma bounds the total expected number of false-positives (both from false-positive candidates and encoding errors) that can be reported by a single query vector (and the vectors in the carry set that it serves as it base).

▶ **Lemma 17.** *The expected number of false positives that are reported when considering all vectors in V_k (whose base represents x_k) as queries is $O(2^{\ell-1}N/R^{1-\frac{1}{\ell}})$.*

Proof. We focus on $v \in V_k$ that is queried when considering x_k. This vector v implies the value of m which is the length of substrings of S that can have v as their Parikh vector. Clearly, there are at most N such substrings. We focus on the substring from location α to location $\alpha + m - 1$ in S. Due to our encoding scheme, this substring contains a (possibly empty) suffix of $ENC_\ell(h(x_i))$, for some x_i, followed by $k-1$ full encodings of some integers from A, and then a (possibly empty) prefix of $ENC_\ell(h(x_j))$, for some integers x_i and x_j. The only way in which we may falsely report location α as a match is if for each $\sigma \in \Sigma$ the number of σ characters in the substring of S, denoted by $f(\sigma, \alpha, m)$, is equal to the count of σ in v, denoted by v_σ. For a given σ, since the substring contains $k-1$ complete encodings, we can consider $v_\sigma - (k-1)R^{1/\ell}$ which is a function of $\bar{p}_{r,h(x_k)}$, compared to $f(\sigma, \alpha, m) - (k-1)R^{1/\ell}$. Now, since $\bar{p}_{r,h(x_k)}$ is uniformly random (due to h) in the range $[R^{1/\ell}]$, the probability that they are equal is $R^{-1/\ell}$. This is true for every character σ on its own, but when considering all of the ℓ characters, once we set the count for the first $\ell-1$ characters the count for the last character completely depends on the other counts. Therefore, the probability that the comparison passes for all of the characters only depends on the first $\ell-1$ characters, and is $1/R^{1-1/\ell}$. By linearity of expectation over all possible locations in S and all $2^{\ell-1}$ vectors in V_k, the expected number of false positives is $O(2^{\ell-1}N/R^{1-\frac{1}{\ell}})$. ◀

7.2 Hardness of Histogram Reporting

Utilizing the reduction we have described in the previous section, that transforms an ordered set A to a string S, we can prove the following CLB.

▶ **Theorem 13** (restated). *Assume the 3SUM conjecture holds. The histogram reporting problem for an N-length string and constant alphabet size $\ell \geq 2$ cannot be solved using $O(N^{2-\frac{2\gamma}{\ell+\gamma}-\Omega(1)})$ preprocessing time, $O(N^{1-\frac{\gamma}{\ell+\gamma}-\Omega(1)})$ query time and $O(N^{\frac{\gamma\ell}{\ell+\gamma}-\frac{2\gamma}{\ell+\gamma}-\Omega(1)})$ reporting time per item, for any $0 < \gamma < \ell$.*

Proof. We follow the reduction in Section 7.1. For an instance of the histogram reporting problem on a string of length N denote the preprocessing time by $O(N^\alpha)$, the query time by $O(N^\beta)$ and the reporting time per item by $O(N^\delta)$. The total expected running time used by our reduction to solve DiffConv3SUM is $O(N^\alpha) + n \cdot O(N^\beta) + E_{fp} \cdot O(N^\delta)$, where E_{fp} is the expected total number of false positives. This running time must be $\Omega(n^{2-\Omega(1)})$, unless 3SUM conjecture is false.

Since $N = O(\ell \cdot R^{\frac{1}{\ell}} n)$ and $E_{fp} = O(n2^\ell N/R^{1-\frac{1}{\ell}})$, then either $(\ell \cdot R^{\frac{1}{\ell}} n)^\alpha = \Omega(n^{2-o(1)})$, $(\ell \cdot R^{\frac{1}{\ell}} n)^\beta = \Omega(n^{1-o(1)})$, or $n2^\ell(\ell \cdot R^{\frac{1}{\ell}} n)/R^{1-\frac{1}{\ell}} \cdot (\ell \cdot R^{\frac{1}{\ell}} n)^\delta = \Omega(n^{2-o(1)})$. Set R to be n^γ. By straightforward calculations following our choice of R we get that $\alpha = 2 - \frac{2\gamma}{\ell+\gamma} - \Omega(1)$, $\beta = 1 - \frac{\gamma}{\ell+\gamma} - \Omega(1)$, and $\delta = \frac{\gamma\ell}{\ell+\gamma} - \frac{2\gamma}{\ell+\gamma} - \Omega(1)$. ◀

7.3 From Reporting to Decision: Hardness of Histogram Indexing

We make use of Theorem 13 to obtain a CLB on the decision variant of the problem. Amir et al. [5] proved similar lower bounds based on a stronger 3SUM conjecture. Our proof here shows that this stronger assumption is not needed and that the common 3SUM conjecture suffices. The idea of the proof is to make the expected number of false-positives small by a suitable choice of R.

▶ **Lemma 18.** *Assume the 3SUM conjecture holds. The histogram indexing problem for a string of length N and constant alphabet size $\ell \geq 3$ cannot be solved with $O(N^{2-\frac{2}{\ell-1}-\Omega(1)})$ preprocessing time and $O(N^{1-\frac{1}{\ell-1}-\Omega(1)})$ query time.*

Proof. We follow the reduction in Section 7.1. In order to use histogram indexing we will reduce the probability of a false positive for any query to be less than 1/2. From Lemma 17 we know that the expected number of false positives due to query is at most $O(\frac{2^{\ell-1}(\ell R^{\frac{1}{\ell}} n)}{R^{1-\frac{1}{\ell}}})$. By setting R to be $c_1 n^{\frac{\ell}{\ell-2}}$ for sufficiently large constant c_1 the number of false positives is strictly smaller than 1/2, which implies immediately that the probability of a false positive is strictly smaller than 1/2. Therefore, if we were to solve histogram indexing instead of histogram reporting on the same input as in Theorem 13, the probability of a false positive is less than 1/2. We can make this probability smaller by repeating the process $O(\log n)$ times, each time using a different hash function h. This way, the probability that all of the queries that are due to a specific x_k return false positives is less than $1/poly(n)$. If a given x_k passes all of the query processes (that is, a positive answer is received by each one of them), then we can verify that there is indeed a match with this x_k in $O(n)$ time, which will add a negligible cost to the expected running time in the case it is indeed a false positive. Thus, the total expected running time of this procedure is $O(\log n(P(N,\ell)+nQ(N,\ell)))$, where $P(N,\ell)$ is the preprocessing time (for input string of length N and alphabet size ℓ) and $Q(N,\ell)$ is the query time (for the same parameters). Therefore, unless the 3SUM conjecture is false, there is no solution for histogram indexing such that $P(N,\ell) = O(n^{2-\Omega(1)})$ and $Q(N,\ell) = O(n^{1-\Omega(1)})$. If we plug-in the value of R we have chosen and follow the calculations in the proof of Theorem 13 (with $\gamma = \frac{\ell}{\ell-2}$), then we obtain that there is no solution for the histogram indexing problem with $P(N,\ell) = O(N^{2-\frac{2}{\ell-1}-\Omega(1)})$ and $Q(N,\ell) = O(N^{1-\frac{1}{\ell-1}-\Omega(1)})$. ◀

We generalize this CLB by presenting a full-tradeoff between preprocessing and query time. The proof will appear in the full version of this paper. The idea of the proof is to artificially split the encoded string S to smaller parts, so we can have many false positives in S, but the probability for a false positive in each part will be small.

▶ **Theorem 14** (restated). *Assume the 3SUM conjecture holds. The histogram indexing problem for a string of length N and constant alphabet size $\ell \geq 3$ cannot be solved with $O(N^{2-\frac{2(1-\alpha)}{\ell-1-\alpha}-\Omega(1)})$ preprocessing time and $O(N^{1-\frac{1+\alpha(\ell-3)}{\ell-1-\alpha}-\Omega(1)})$ query time, for any $0 \leq \alpha \leq 1$.*

References

1 Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In *Int'l Colloquium on Automata, Languages and Programming, ICALP 2013*, pages 1–12, 2013.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science, FOCS 2014*, pages 434–443, 2014.

3 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages and Programming, ICALP 2014*, pages 39–51, 2014.

4 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Symposium on Theory of Computing, STOC 2015*, pages 41–50, 2015.

5 Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *International Colloquium on Automata, Languages and Programming, ICALP 2014*, pages 114–125, 2014.

6 Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. In *Workshop on Algorithms and Data Structures, WADS 2005*, pages 409–421, 2005.

7 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014.

8 Peter Burcsi, Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Algorithms for jumbled pattern matching in strings. *Int. J. Found. Comput. Sci.*, 23(2):357–374, 2012.

9 Timothy Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Symposium on Theory of Computing, STOC 2015*, pages 31–40, 2015.

10 Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Searching for jumbled patterns in strings. In *Prague Stringology Conference*, pages 105–117, 2009.

11 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Symposium on Theory of Computing, STOC 2002*, pages 592–601, 2002.

12 Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Symposium on Theoretical Aspects of Computer Science, STACS 1996*, pages 569–580, 1996.

13 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

14 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 2014*, pages 296–303, 2014.

15 Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse fourier transform. In *Symposium on Theory of Computing Conference, STOC 2012*, pages 563–578, 2012.

16 Danny Hermelin, Gad M. Landau, Yuri Rabinovich, and Oren Weimann. Binary jumbled pattern matching via all-pairs shortest paths. *CoRR*, abs/1401.2065, 2014.

17 Allan Grønlund Jørgensen and Seth Pettie. Threesomes, degenerates, and love triangles. In *Foundations of Computer Science, FOCS 2014*, pages 621–630, 2014.

18 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Efficient indexes for jumbled pattern matching with constant-sized alphabet. In *ESA*, pages 625–636, 2013.

19 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *Symposium on Discrete Algorithms, SODA 2016*, pages 1272–1287, 2016.

20 Tanaeem M. Moosa and M. Sohel Rahman. Indexing permutations for binary strings. *Inf. Process. Lett.*, 110(18-19):795–798, 2010.

21 Tanaeem M. Moosa and M. Sohel Rahman. Sub-quadratic time and linear space data structures for permutation matching in binary strings. *J. Discrete Algorithms*, 10:5–9, 2012.

22 S. Muthukrishnan. New results and open problems related to non-standard stringology. In *CPM 1995*, pages 298–317, 1995.

23 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Symposium on Theory of Computing Conference, STOC 2010*, pages 603–610, 2010.

24 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014*, pages 664–673, 2014.

25 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Symposium on Theory of Computing Conference, STOC 2012*, pages 887–898, 2012.