Sub-string/Pattern Matching in Sub-linear Time Using a Sparse Fourier Transform Approach

Nagaraj T. Janakiraman, Avinash Vem, Krishna R. Narayanan, Jean-Francois Chamberland
Department of Electrical and Computer Engineering
Texas A&M University
{tjnagaraj,vemavinash,krn,chmbrlnd}@tamu.edu

ABSTRACT

We consider the problem of querying a string (or, a database) of length N bits to determine all the locations where a substring (query) of length M appears either exactly or is within a Hamming distance of K from the query. We assume that sketches of the original signal can be computed off line and stored. Using a sparse Fourier transform computation based approach, we show that all such matches can be determined with high probability in sub-linear time. Specifically, if the query length $M = O(N^{\mu})$ and the number of matches $L = O(N^{\lambda})$, we show that for $\lambda < 1 - \mu$ all the matching positions can be determined with a probability that approaches 1 as $N \to \infty$ for $K \leq \frac{1}{6}M$. More importantly our scheme has a worst-case computational complexity that is only $O\left(\max\{N^{1-\mu}\log^2 N, N^{\mu+\lambda}\log N\}\right)$, which means we can recover all the matching positions in *sub-linear* time for $\lambda < 1 - \mu$. This is a substantial improvement over the best known computational complexity of $O\left(N^{1-0.359\mu}\right)$ for recovering one matching position by Andoni *et al.* [2]. Further, the number of Fourier transform coefficients that need to be computed, stored and accessed, i.e., the sketching complexity of this algorithm is only $O\left(N^{1-\mu}\log N\right)$. Several extensions of the main theme are also discussed.

KEYWORDS

Pattern Matching, String Matching, Sub-linear time Algorithms, Sparse Signal Processing, Hashing, Sparse Fourier Transform, Compressed Sensing

1 INTRODUCTION AND PROBLEM STATEMENT

We consider the substring/pattern matching problem, which has been studied extensively in theoretical computer science. In this problem, a signal $\vec{\mathrm{x}} := (x[0], x[1], \ldots, x[N-1])$ of length N symbols representing a string, library, or database is available. The objective is to answer queries regarding whether a given string $\vec{\mathrm{y}} := (y[0], y[1], \ldots, y[M-1])$ of length M is a substring of $\vec{\mathrm{x}}$. We are especially interested in the case where a sketch of $\vec{\mathrm{x}}$ can be computed offline and stored, and where the one-time computational complexity of creating the sketch can be amortized over repeated queries or ignored. Moreover, we focus on the random setting in which the $x[i]$'s form a sequence of independent and identically distributed (i.i.d.) random variables, each taking values in $\mathcal{A} \subset \mathbb{R}$. We begin our analysis by restricting our attention to the binary case where $\mathcal{A} := \{\pm 1\}$.[1] Within this context, we consider the following two settings:

Exact Pattern Matching: In the exact pattern matching problem, a substring of length M of $\vec{\mathrm{x}}$, namely $\vec{\mathrm{y}} := \vec{\mathrm{x}}[\tau : \tau + M - 1]$, is obtained by taking M consecutive symbols from $\vec{\mathrm{x}}$ and is presented as a query. This pattern may appear within $\vec{\mathrm{x}}$ in up to L different locations $\tau_1, \tau_2, \ldots, \tau_L$ and the task is to determine all the locations $\tau_i, \forall i \in [1 : L]$. We consider the probabilistic version where our objective is to recover the matching locations with a probability that approaches 1 as $M, N \to \infty$.

Approximate Pattern Matching: In approximate pattern matching, $\vec{\mathrm{y}}$ is a noisy version of a substring, i.e., $\vec{\mathrm{y}} = \vec{\mathrm{x}}[\tau : \tau + L - 1] \odot \vec{\mathrm{b}}$, where $\vec{\mathrm{b}}$ is a noise sequence with $b[i] \in \{\pm 1\}$ such that $d_H(\vec{\mathrm{y}}, \vec{\mathrm{x}}[\tau : \tau + M - 1]) \leq K$. Function d_H denotes the Hamming distance, and $\odot$ represents component-wise multiplication. The objective is to determine all locations τ_i such that $d_H(\vec{\mathrm{y}}, \vec{\mathrm{x}}[\tau_i : \tau_i + M - 1]) \leq K$ with a probability that approaches 1 as K, M and $N \to \infty$.

We evaluate our proposed algorithm according to two metrics - (i) the space required to store the sketch of $\vec{\mathrm{x}}$, which we refer to as sketching complexity, and (ii) the computational complexity in answering the query.

These problems are relevant to many applications including text matching, audio/image matching, DNA matching in genomics, metabolomics, radio astronomy, searching for signatures of events within large databases, etc. The proposed techniques are particularly relevant now due to the interest in applications involving huge volumes of data. Our proposed approach is most useful in the following situations. (i) The string $\vec{\mathrm{x}}$ is available before querying and one time computations such as computing a sketch of $\vec{\mathrm{x}}$ can be performed offline and stored, and the complexity of computing the sketch of $\vec{\mathrm{x}}$ can be ignored. Then, when queries in the form of $\vec{\mathrm{y}}$ appear, one would like to decrease the computational complexity in searching for the string $\vec{\mathrm{y}}$. (ii) The string $\vec{\mathrm{x}}$ is not centrally available, but parts of the string are sensed by different data collecting nodes distributively and communicated to a central server. A search query is presented to the server; and this server must decide whether the string appears in the data sensed by one or more of the distributed nodes and, if present, it must also identify when the queried string appeared. In this latter case, we wish to minimize the

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

[1] Extensions to other alphabets $\mathcal{A} \subset \mathbb{R}$ are straightforward. Extension to the non-i.i.d. case is also possible.

amount of data communicated by the nodes to the server and the computational complexity in searching for the string. The proposed formulation is most useful when the query $\vec{y}$ is not a pattern that can be predicted and, therefore, creating a lookup table to quickly identify commonly-occurring patterns will not be effective.

A naive approach to searching for substring $\vec{y}$ in $\vec{x}$ is to first compute the cross-correlation between $\vec{x}$ and $\vec{y}$, which we denoted by $\vec{r} = [r[0], r[1], \ldots, r[N-1]]$ with

$$r[m] = (\vec{x} * \vec{y})[m] \triangleq \sum_{i=1}^{M} x[m+i-1]y[i], \tag{1}$$

and, subsequently, choosing the index m that maximizes $r[m]$. This strategy uses the N samples of $\vec{x}$ and has a super-linear computational complexity of $MN = O(N^{1+\mu})$. A computationally more efficient approach uses the fact that $\vec{r}$ can be computed by taking the inverse Fourier transform of the product of the Fourier transforms of $\vec{x}$ and $\vec{y}^*[-n]$, where $\vec{y}^*[-n]$ is the conjugate of the time reversed version of $\vec{y}$. Such an approach still uses all the N samples of $\vec{x}$, but reduces the computational complexity to $O(N \log N)$. Note that even though the Fourier transform of $\vec{x}$ can be precomputed, the N-point Fourier transform of $\vec{y}$ still needs to be computed online resulting in the $O(N \log N)$ computational complexity.

Both the exact pattern matching problem and the approximate pattern matching problem have been extensively studied in computer science. Three recent articles [2], [1] and [11] provide a brief summary of existing contributions. For the exact matching problem, the Rabin-Karp algorithm solves a more general problem of finding the occurrence of a set of query strings. However, the algorithm has a computational complexity that is at least linear in N. Boyer and Moore presented an algorithm in [3] for finding the first occurrence of the match (only τ_1) that has an expected complexity of $O(N/M \log N) = O(N^{1-\mu} \log N)$, whereas the worst case complexity (depending on τ_1's) can be $O(N \log N)$. For large M, the algorithm indeed has an average complexity that is sub-linear in N. More recently, it has been shown that techniques based on the Burrows-Wheeler transform can be used to solve the exact matching problem with sub-linear time complexity [5] using a storage space of $O(N)$ bits. This problem is well studied under the read alignment setting by the Bioinformatics community [9, 10]. For small $|\mathcal{A}|$, it has the best known complexity; however, the complexity increases with $|\mathcal{A}|$. Further, extensions to approximate matching setting [13] has a complexity that increases exponentially in K and, hence, appear to be infeasible for $K = O(M)$. The Boyer and Moore algorithm has been generalized to the approximate pattern matching problem in [4] with an average case complexity of $O(NK/M \log N)$, which provides a sub-linear time algorithm only when $K \ll M$. In [2], Andoni, Hassanieh, Indyk and Katabi have given the first sub-linear time algorithm with a complexity of $O(N/M^{0.359})$ for $K = O(M)$.

2 OUR MAIN RESULTS AND RELATION TO PRIOR WORK

Assume that a sketch of $\vec{x}$ of size $O(\frac{N}{M} \log N)$ can be computed and stored. Then for the *exact pattern matching* and *approximate pattern matching* (with $K = \eta M$) problems, with the number of matches L scaling as $O(N^{\lambda})$, we show an algorithm that has

- a sketching function for $\vec{y}$ that computes $O(\frac{N}{M} \log N) = O\left(N^{1-\mu} \log N\right)$ samples
- a computational complexity of $O\left(\max\{N^{1-\mu} \log^2 N, N^{\mu+\lambda} \log N\}\right)$
- a decoder that recovers all the L matching positions with a failure probability that approaches zero asymptotically in N

When $L < O\left(\frac{N}{M}\right)$ (i.e. $\lambda < 1-\mu$), which is typically the interesting case, our algorithm has a *sub-linear time and space complexity*.

Consider the following two real world examples to appreciate the impact our results.

Example 2.1. Let us consider the case of time series data, an audio signal which is sampled at a moderate sampling rate of 10 KHz and let the signal $\vec{x}$ correspond to 1 hour of audio and hence, $N = 3.6 \times 10^7$. Let the query $\vec{y}$ be a 10 second clip which corresponds to $M = 10^4$ resulting in a $\mu = 0.529$. Even though the time duration of $\vec{y}$ is only 0.3% of the duration of $\vec{x}$, the 10KHz sampling results in a fairly large $\mu = 0.529$. This will result in the complexity of the algorithm being only proportional to $\sqrt{N}$.

Example 2.2. Suppose a database of genomes contains a sequence $\vec{x}$ of length $N = 10^{12}$. Consider a metabolomics application where we perform partial reads of DNA sequences of organisms in an environment, and we wish to query whether this organism appears in the database. With current sequencing technology, we can read DNA sequences of length $M = 1000$ to $M = 100000$ in length (e.g., Oxford Nanopore technology can provide long reads) and the reads are typically in error with a small probability. Naively searching for the DNA sequence within the database would entail computations on the order of $10^{15}-10^{17}$ operations. Even linear complexity would require computations on the order of 10^{12} operations. Our proposed technique provide a nearly 200 times reduction in complexity over linear-time algorithms for these parameters, thereby bridging the gap from theoretically possible to practical.

There are important differences between our paper and [1–3, 6]. First and foremost, the algorithms used for pattern matching are entirely different. While their algorithms are combinatorial in nature, our algorithm is algebraic and uses signal processing and coding theoretic primitives. Secondly, the system model considered in our paper differs from the model in [1–3, 6] in that we allow for preprocessing or creating a sketch of the data $\vec{x}$. Our algorithm exploits this fact and results in a computational complexity $O(N/M)$ which is better than that in [2] for the approximate pattern matching problem. Finally, we also consider the problem of finding all matches of the pattern $\vec{y}$ instead of looking for only one match.

Our paper is inspired by and builds on two recent works by Hassanieh *et al.* in [6] and Pawar and Ramchandran [12]. In [6], Hassanieh *et al.*, considered the correlation function computation problem for a Global Positioning System (GPS) receiver and exploited the fact that the cross-correlation vector $\vec{r}$ is a very sparse signal in the time domain and, hence, the Fourier transform of $\vec{r}$ need not be evaluated at all the N points. In the GPS application, which was the focus of [6], the query $\vec{y}$ corresponds to the received signal from the satellites and, hence, the length of the query was at least N. As a result, the computational complexity is still $O(N \log N)$ (still linear in N) and only the constants were improved in relation to

the approach of computing the entire Fourier transform. In a later paper by Andoni *et al.,* [2], a sub-linear time algorithm for shift finding in GPS is presented; however, this algorithm is completely combinatorial and eschews algebraic techniques such as FFT-based techniques.

In [12], Pawar and Ramchandran present an algorithm based on aliasing and the peeling decoder for computing the Fourier transform of a signal with noisy observations for the case when the Fourier transform is known to be sparse. This algorithm has a complexity of $O(N \log N)$ (near-linear in N) and they do not consider the pattern matching problem. Our algorithm follows a similar approach to that of Pawar and Ramchandran's algorithm with one important distinction. We modify their algorithm to exploit the fact that the peak of the correlation function of interest is always positive. This modification allows us to compute the Sparse Inverse Discrete Fourier Transform (SIDFT) with sub-linear time complexity of $O(N^{1-\alpha} \log N)$, $0 < \alpha \leq 1$ as opposed to a near-linear complexity . One of the main contributions of this paper is to show that signal processing and coding theoretic primitives, i.e., Pawar and Ramchandran's algorithm with key modifications can be used to solve the pattern matching algorithm in sub-linear time.

3 NOTATIONS

The table below introduces the notations we adopt throughout this paper. We denote signals and vectors using the standard vector notation of arrow over the letter, time domain signals using lowercase letters and the frequency domain signals using uppercase letters. For example $\vec{\mathrm{x}} = (x[0], x[1], \ldots, x[N-1])$ denotes a time domain signal with i^{th} time component denoted by $x[i]$, and $\vec{X} = \mathcal{F}_N\{\vec{\mathrm{x}}\}$ denotes the N-point Fourier coefficients of $\vec{\mathrm{x}}$. We denote matrices using boldface upper case letters. We denote the set $\{0, 1, 2 \cdots, N-1\}$ by $[N]$.

Symbol	*Meaning*
N	Size of the string or database in symbols
M	Length of the query in symbols
L	Number of matches
μ	Smallest $0 < \mu < 1$ such that $M = O(N^{\mu})$
λ	Smallest $0 < \lambda < 1$ such that $L = ON^{\lambda})$
K	$\max_{\tau} d_H(\vec{\mathrm{x}}[\tau : \tau + M - 1], \vec{\mathrm{y}})$
η	$\frac{K}{M}$
d	Number of stages in the FFAST algorithm
$f_i \approx N^{\alpha}$	Length of smaller point IDFT at each stage-i
$g_i = N/f_i$	Sub-sampling factor in Fourier domain for stage-i
B	Number of shifts (or branches) in each stage
$G = N^{\gamma}$	Number of blocks (for parallel processing)
$\tilde{N} = N^{1-\gamma}$	Length of one block (for parallel processing)

Table 1: Parameters and various quantities involved in describing the algorithm

4 DESCRIPTION OF THE ALGORITHM

In this section, given the input string $\vec{\mathrm{x}}$ and the query string $\vec{\mathrm{y}}$, we describe our algorithm that finds the matching positions $\mathcal{T} := \{\tau_1, \tau_2, \cdots \tau_L\}$ with sample and time complexities that are sub-linear in N. The main idea exploits the fact that the correlation vector $\vec{\mathrm{r}}$ is sparse (upto some noise terms) with dominant peaks at L matching positions denoted by $\mathcal{T}$ and noise components at $N - L$ positions where the strings do not match.

Consider the correlation signal $\vec{\mathrm{r}}$ in the case of exact matching:

$$r[m] = \begin{cases} M, \text{ if } m \in \mathcal{T} \\ n_m, \ m \in [N] - \mathcal{T} \end{cases} \tag{2}$$

where n_m is the noise component that is induced due to correlation of two i.i.d. sequence of random variables each taking values from $\mathcal{A} := \{+1, -1\}$. The sparse vector $\vec{\mathrm{r}}$ can be computed indirectly using Fourier transform approach as shown below:

$$\vec{\mathrm{r}} = \underset{\text{III}}{\mathcal{F}_N^{-1}} \{\underset{\text{I}}{\mathcal{F}_N\{\vec{\mathrm{x}}\}} \odot \underset{\text{II}}{\mathcal{F}_N\{\vec{\mathrm{y}}'\}}\} \tag{3}$$

where $\mathcal{F}_N\{\cdot\}$ and $\mathcal{F}_N^{-1}\{\cdot\}$ refer to N-point discrete Fourier transform and its inverse respectively, $\odot$ is the point-wise multiplication operation and $y'[n] = y^*[-n]$. Fig. 1 presents a notional schematic of our Algorithm. As evident from Eq. (3), our algorithm for computing $\vec{\mathrm{r}}$ consists of three stages:

- Computing the sketch $\vec{X} = \mathcal{F}_N\{\vec{\mathrm{x}}\}$ of $\vec{\mathrm{x}}$
- Computing the sketch $\vec{Y}' = \mathcal{F}_N\{\vec{\mathrm{y}}'\}$ of $\vec{\mathrm{y}}$
- Computing the IDFT of $\vec{R} = \vec{X} \odot \vec{Y}'$ given $\vec{X}$ and $\vec{Y}'$

4.1 Sparse Inverse Discrete Fourier Transform

In this section we present Robust Sparse Inverse Discrete Fourier Transform(RSIDFT) scheme that exploits sparsity in the cross-correlation signal $\vec{\mathrm{r}}$ and efficiently recovers its L dominant coefficients. The architecture of RSIDFT is similar to that of the FFAST scheme proposed in [12], but the decoding algorithm has some modifications to handle the noise model induced in this problem. We will see in Sec. 4.2 how the sketches $\vec{X}$ and $\vec{Y}'$ are computed efficiently but for this section we will focus only on the recovery of the sparse coefficients in $\vec{\mathrm{r}}$ given $\vec{X}$ and $\vec{Y}'$.

Consider the RSIDFT framework shown in Fig. 2. Let $\vec{R} = \vec{X} \odot \vec{Y}'$ be the DFT of the cross-correlation signal of $\vec{\mathrm{x}}$ and $\vec{\mathrm{y}}$. We begin by factoring N into d relatively prime factors $\{f_1, f_2, \ldots, f_d\}$, where d is a parameter in the algorithm. The design scheme for choosing f_i's for various values of μ such that f_i divides N and $f_i = N^{\alpha} + O(1) \forall\ i \in [d]$ are given in Sec. 4.1.2. The RSIDFT algorithm consists of *d-stages* with each stage corresponding to a sub-sampling factor of $\frac{N}{f_i}$. In each stage, there are $B = O(\log N)$ *branches* with shifts from the set $\{s_1, s_2, \cdots s_B\}$, where $s_1 = 0$ in the first branch and the rest are chosen uniformly at random from $[N]$.

Given the input $\vec{R}$, in branch j of i^{th} stage of RSDIFT, referred to as *branch* (i, j) for simplicity, RSIDFT sub-samples the signal $\vec{R}$ at

$$\mathcal{S}_{i,j} := \{s_j,\ s_j + g_i,\ s_j + 2g_i,\ \cdots s_j + (f_i - 1)g_i\}, \quad i \in [d], j \in [B] \tag{4}$$

where $g_i := \frac{N}{f_i}$ to obtain $\vec{R}_{i,j} := \vec{R}[\mathcal{S}_{i,j}]$. The sub-sampling operation is followed by a f_i-point IDFT in *branch (i,j)* of stage-i to obtain $\vec{\mathrm{r}}_{i,j}$. Notice that $\vec{\mathrm{r}}_{i,j}$ is an aliased version of $\vec{\mathrm{r}}$ due to the property that sub-sampling in Fourier domain is equivalent to aliasing in time domain.

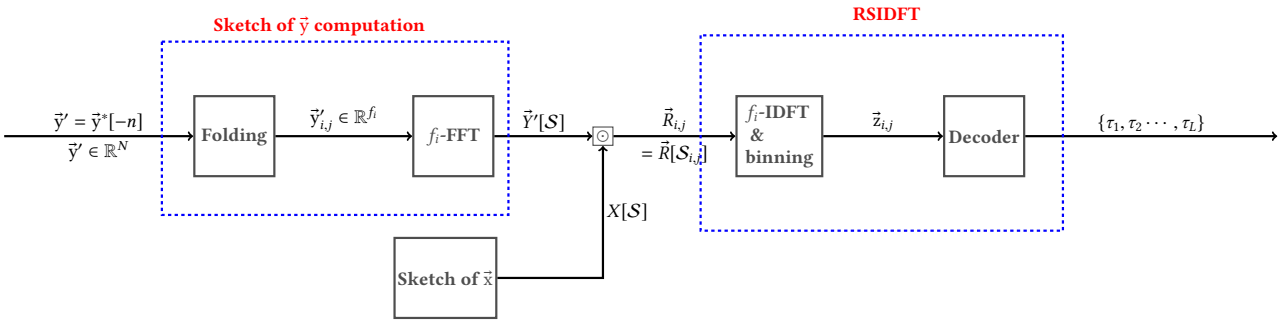


Figure 1: Schematic of the proposed scheme using sparse Fourier transform computation.

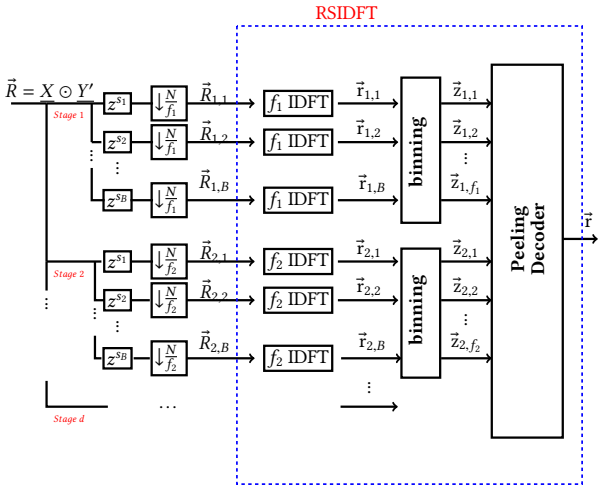


Figure 2: RSIDFT Framework to compute inverse Fourier Transform of a signal $\vec{R}$ that is sparse in time domain.

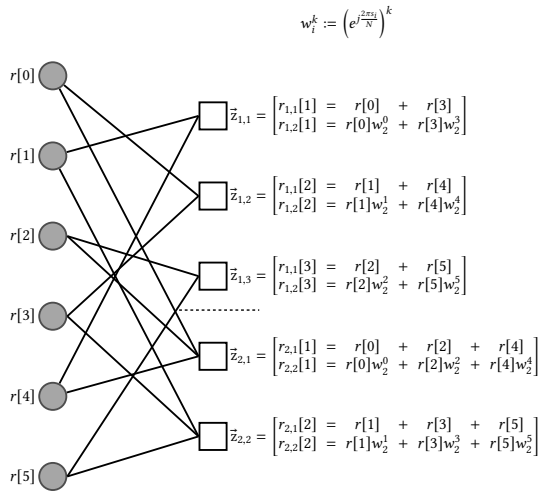


Figure 3: Example of a Tanner graph formed in a RSIDFT framework with system parameters being $N = 6$, $f_1 = 2$, $f_2 = 3$ (i.e., $d = 2$) and $B = 2$. The variable nodes (gray circles) represent the cross-correlation vector $\vec{r}$ and the bin nodes (white squares) represent the binned observation vector $\vec{z}_{i,k}$. The figure illustrates the relationship between $\vec{z}_{i,k}$ and $\vec{r}$.

Let $\vec{z}_{i,k} \in \mathbb{R}^B$, for $k \in [f_i] + 1$, be the kth *binned* observation vector of stage-i formed by stacking $\vec{r}_{i,j}[k], j \in [B]$, together as a vector i.e.

$$\vec{z}_{i,k} = \left[r_{i,1}[k],\ r_{i,2}[k],\ \cdots,\ r_{i,B}[k]\right]^T$$

Note that this gives us a total of f_i binned observation vectors in each stage-i. Using the properties of Fourier transform, we can write the relationship between the observation vectors $\vec{z}_{i,k}$ at bin (i,k) and sparse vector to be estimated $\vec{r}$ as:

$$\vec{z}_{i,k} = \mathbf{W}_{i,k} \times \left[r[k],\ r[k+f_i],\ \cdots,\ r[k+(g_i-1)f_i]\right]^T \quad (5)$$

where we refer to $\mathbf{W}_{i,k}$ as the sensing matrix at bin (i,k) and is defined as

$$\mathbf{W}_{i,k} = \left[\vec{w}^k, \vec{w}^{k+f_i}, \ldots, \vec{w}^{k+(g_i-1)f_i}\right] \quad (6)$$

and $\vec{w}^k = \left[e^{\frac{j2\pi k s_1}{N}},\ e^{\frac{j2\pi k s_2}{N}},\ \cdots,\ e^{\frac{j2\pi k s_B}{N}}\right]^T$

We represent the relation between the set of observation vectors $\{\vec{z}_{i,k}, i \in [1:d], k \in [f_i]\}$ and $\vec{r}$ using a Tanner graph, an example of which is shown in Fig. 3. The nodes on the left, which we refer to as *variable nodes*, represent the N elements of vector $\vec{r}$. Similarly the nodes on the right, which we refer to as *bin nodes*, represent the $\sum_{i \le d} f_i$ sub-sensing signals. We will now describe the decoding algorithm which takes the set of observation vectors $\{\vec{z}_{i,k}, i \in [1:d], k \in [f_i]\}$, each of length B, at df_i bins as input and estimates the L-sparse $\vec{r}$.

4.1.1 Decoder. Observe from the Tanner graph that the degree of each variable node is d and that of each bin node at stage i is g_i. A variable node is referred to as non-zero if it corresponds to a matching position and as zero if it corresponds to a non-matching position. Note that even though the cross-correlation vector value corresponding to a non-matching position is not exactly zero but some negligible noise value we refer to them as zero variable nodes for simplicity. We refer to a bin node as *zero-ton* (or $\mathcal{H}_z$) if the number of non-zero variable nodes connected to the bin node is zero. The *singleton* ($\mathcal{H}_s$), *double-ton* ($\mathcal{H}_d$) *and multi-ton* ($\mathcal{H}_m$) bin nodes are defined similarly where the number of non-zero variable nodes connected are one, two and greater than two, respectively. The peeling decoder has the following three steps in the decoding process.

Bin Classification. In this step a bin node is classified either as a zero-ton or a singleton or a multi-ton. At bin (i, j) the classification is done based on comparing the first observation $z_{i,j}[1]$, which corresponds to zero shift, with a predefined threshold. For $z_{i,j}[1] = z$, the classification hypothesis at bin (i, j), $\widehat{\mathcal{H}}_{i,j}$, can be written as follows:

$$\widehat{\mathcal{H}}_{i,j} = \begin{cases} \mathcal{H}_z & z/M < \gamma_1 \\ \mathcal{H}_s & \gamma_1 < z/M < \gamma_2 \\ \mathcal{H}_d & \gamma_2 < z/M < \gamma_3 \\ \mathcal{H}_m & z/M > \gamma_3 \end{cases} \tag{7}$$

where $(\gamma_1, \gamma_2, \gamma_3) = (\frac{1-2\eta}{2}, \frac{3-4\eta}{2}, \frac{5-6\eta}{2})$. Note that for the case of exact matching $\eta = K/M = 0$.

Singleton decoding. If a bin node (i, j) is classified as a singleton in the bin classification step, we need to identify the position of the non-zero variable node connected to it. This is done by correlating the observation vector $\vec{z}_{i,j}$ with each column of the sensing matrix $\mathbf{W}_{i,j} := [\vec{w}^j, \vec{w}^{j+f_i}, \ldots, \vec{w}^{j+(g_i-1)f_i}]$ and choosing the index that maximizes the correlation value.

$$\hat{k} = \underset{k \in \{j+lf_i\}}{\arg\max}\ \vec{z}_{i,j}^{\dagger} \vec{w}^k$$

where † denotes the conjugate transpose. The value of the variable node connected to the singleton bin is decoded as:

$$\hat{r}[\hat{k}] = M(1-\eta).$$

Note that for the case of exact matching we know the value to be exactly equal to M. But in the case of approximate matching, the actual value of $r[k] \in \{M(1-2\eta), \ldots, M-1, M\}$ and our estimate $\hat{r}[\hat{k}] = M(1-\eta)$ is only approximate. But this suffices for recovering the positions of matches i.e., the indices of the sparse coefficients in $\vec{r}$.

Peeling Process. The peeling based decoder we employ consists of finding a singleton bin, then identifying the single non-zero variable node connected to the bin, decoding its value and removing (*peeling off*) it's contribution from all the bin nodes connected to that variable node. The main idea behind this decoding scheme is that (for appropriately chosen parameters), at each iteration, peeling a singleton node off will induce at least one more singleton bin and the process of peeling off can be repeated iteratively. Although the main idea is similar for exact matching and the approximate matching scenarios, there are some subtle differences in their implementation.
Exact Matching: In the case of exact matching, we remove the decoded variable node's contribution from all the bin nodes it is connected to.
Approximate Matching: In this case, similar to the approach in [8], we remove the decoded variable node's contribution only from bins that are originally a singleton or a double-ton. We do not alter the bins which are classified to be multi-tons with degree more than two.

Note: The differences in peeling implementation for exact and approximate matching cases is because unlike exact matching the approximate matching may result in non-zero error (e_1) between the actual correlation value $r[k]$ and the estimate $\hat{r}[\hat{k}]$, i.e. $e_1 = |r[k] - \hat{r}[\hat{k}]|$ with $0 \le e_1 \le \eta M$. This may lead to error propagation if we use the same decoding scheme as in exact matching. Hence to overcome this problem we impose a constraint on the type of bin nodes participating in the peeling process.

We present the overall recovery algorithm, which comprises of *bin classification, singleton decoding* and *peeling process*, in the Algorithm.1 pseudo-code. Note that $\mathcal{N}(k)$ denote the neighborhood for variable node k i.e., the set of bins connected to k^{th} variable node.

Algorithm 1 Peeling based recovery algorithm

Compute $\widehat{\mathcal{H}}_{i,j}$ for $i \in [d], j \in [f_i]$. (See Eqn. (7))
while $\exists\ i, j : \widehat{\mathcal{H}}_{i,j} = \mathcal{H}_s$, **do**
  $(\hat{k}, \hat{r}[\hat{k}])$ =**Singleton-Decoder**$(\vec{z}_{i,j})$
  Assign $\hat{r}[\hat{k}]$ to $\hat{k}^{\text{th}}$ variable node
  for $(i_0, j_0) \in \mathcal{N}(\hat{k})$ **do**
    if *Exact Matching* ***then***
      $\vec{z}_{i_0,j_0} \leftarrow \vec{z}_{i_0,j_0} - \hat{r}[\hat{k}]\vec{w}^{\hat{k}}$
    else
      $\vec{z}_{i_0,j_0} \leftarrow \vec{z}_{i_0,j_0} - \hat{r}[\hat{k}]\vec{w}^{\hat{k}}$ *only if* $\widehat{\mathcal{H}}_{i_0,j_0} = \mathcal{H}_s$ *or* $\mathcal{H}_d$
    Re-do the bin classification for (i_0, j_0) *and compute* $\widehat{\mathcal{H}}_{i_0,j_0}$

Algorithm 2 Singleton-Decoder

Input: $\vec{z}_{i,j}$
Output: $(\hat{k}, \hat{r}[\hat{k}])$
Estimate singleton index to be $\hat{k} = \underset{k \in \{j+lf_i\}}{\arg\max}\ \vec{z}_{i,j}^{\dagger} \vec{w}^k$
Estimate the value to be:

$$\hat{r}[\hat{k}] = \begin{cases} M & \text{Exact Matching case} \\ M-K & \text{Approximate Matching case} \end{cases}$$

4.1.2 Choosing f_i and α for various μ. For a given value of μ, we will describe how to choose the parameters d and f_i. Find a factorization for signal length $N = \prod_{i=0}^{d-1} P_i$ such that the set of integers $\{P_0, P_1, \ldots, P_{d-1}\}$ are pairwise co-prime and all the P_i are approximately equal. More precisely, let $P_i = \mathbf{F} + O(1)\ \forall i$ for some value $\mathbf{F}$. We can add zeros at the end of the vector $\vec{x}$ and increase the length of the vector until we are able to find a factorization that satisfies this property.

- For $\mu < 0.5$: Choose $f_i = N/P_i$.
  Exact Matching: Find $d \in \mathbb{N}\backslash\{1, 2\}$ such that $\mu \in (\frac{1}{d}, \frac{1}{d-1}]$
  Approximate Matching: If $\mu \in (\frac{1}{8}, \frac{1}{2})$, choose $d = 8$. Else find $d \ge 8$ such that $\mu \in (\frac{1}{d}, \frac{1}{d-1})$

- For $\mu > 0.5$: Choose $f_i = P_i$
  Exact Matching: Find $d \in \mathbb{N}\backslash\{1, 2\}$ such that $\mu \in (1 - \frac{1}{d-1}, 1 - \frac{1}{d}]$
  Approximate Matching: If $\mu \in (\frac{1}{2}, \frac{1}{8})$, choose $d = 8$. Else find $d \ge 8$ such that $\mu \in (1 - \frac{1}{d-1}, 1 - \frac{1}{d})$

Thus, for both the exact and approximate matching cases, for any $0 < \mu < 1$, we choose the down-sampling factors f_i to be approximately equal to N^α where $\alpha > 1 - \mu$.

4.1.3 Distributed processing framework. Given a database(or string) of length N, we divide the database into $G = N^\gamma$ blocks each of length $\tilde{N} = N/G$. Now each block can be processed independently (in parallel) using the RSIDFT framework with the new database length reduced from N to $\tilde{N}$. This distributed framework has the following advantages

- Firstly, this enables parallel computing and hence can be distributed across different workstations.
- Improves the sample and computational complexity by a constant factor.
- Sketch of the database needs to be computed only for a smaller block length and hence requires computation of only a shorter $\tilde{N}$ point FFT.
- Helps overcome implementation issues with memory and precision as the scale of the problem is reduced.

4.2 Sketches of $\vec{X}$ and $\vec{Y}$

As we have already seen in Sec. 4.1 the RSIDFT framework requires the values of $\vec{R}(= \vec{X} \odot \vec{Y})$ only at indices $\mathcal{S}$ or in other words we need $\vec{X}$ and $\vec{Y}$ only at the indices in set $\mathcal{S}$ of cardinality dBf_i. We assume that the sketch of $\vec{\mathrm{x}}$, $\vec{X}[\mathcal{S}] = \{X[i], i \in \mathcal{S}\}$ is pre-computed and stored in a database.

Computing the sketch of $\vec{\mathrm{y}}$. : For every new query $\vec{\mathrm{y}}$, only $\{\vec{Y}'[\mathcal{S}_{i,j}], i \in [d], j \in [B]\}$ needs to be computed where the subsets $\mathcal{S}_{i,j}$, defined in Eq. (4), are

$$\mathcal{S}_{i,j} := \{ s_j + kg_i : k \in [f_i]\}, \quad i \in [d], j \in [B] \tag{8}$$

of cardinality f_i. Naively, the FFT algorithm can be used to compute N-pt DFT of $\vec{Y}'$ and the required subset of coefficients can be taken but this is inefficient and would be of $O(N \log N)$ complexity. Instead, we observe that $\vec{Y}'[\mathcal{S}_{i,j}]$ is $\vec{Y}'$ shifted by s_j and sub-sampled by a factor of g_i. Thus for a given (i, j) this corresponds to, in time domain, a point-wise multiplication by $[1, w_{s_j}, w_{s_j}^2, \ldots, w_{s_j}^{N-1}]$ followed by *folding* the signal into $g_i(= \frac{N}{f_i})$ signals each of length f_i and adding them up resulting in a single length-f_i signal denoted by $\vec{\mathrm{y}}'_{i,j}$. Formally the *folding* operation can be described as follows:

$$\vec{\mathrm{y}}'_{i,j} = \sum_{m=0}^{g_i-1} \vec{\mathrm{y}}'[mf_i : (m+1)f_i - 1] \odot [w_{s_j}^{mf_i}, w_{s_j}^{mf_i+1}, \ldots, w_{s_j}^{(m+1)f_i-1}],$$

where, $w_{s_j} = e^{-\frac{j2\pi s_j}{N}}$. Taking f_i-point DFT of $\vec{\mathrm{y}}'_{i,j}$ produces $\vec{Y}'[\mathcal{S}_{i,j}]$ i.e.

$$\vec{Y}'[\mathcal{S}_{i,j}] = \mathcal{F}_{f_i}\{\vec{\mathrm{y}}'_{i,j}\}$$

which is what we need in branch (i, j). To obtain all the samples in $\mathcal{S}$ required for the RSDIFT framework, the *folding* technique followed by a DFT needs to be carried out for each (i, j), for $i \in [d], j \in [B]$, a total of dB times N^α-point DFT.

5 PERFORMANCE ANALYSIS

In this section, we will analyze the overall probability of error involved in finding the correct matching positions. This can be done by analyzing the following three error events independently and then using a union bound to bound the total probability of error.

- $\mathcal{E}_1$-*Bin Classification*: Event that a bin is wrongly classified.
- $\mathcal{E}_2$-*Position Identification*: Given a bin is correctly identified as a singleton, event that the position of singleton is identified incorrectly.
- $\mathcal{E}_3$-*Peeling Process*: Given the classification of all the bins and the position identification of singletons in each iteration is accurate, event that the peeling process fails to recover the L significant correlation coefficients.

5.1 Bin Classification

Lemma 5.1. *The probability of bin classification error at any bin (i, j) can be upper bounded by*

$$\mathbb{P}[\mathcal{E}_1] \leq 6e^{-\frac{N^{\mu+\alpha-1}(1-6\eta)^2}{16}}$$

Proof.

$$\begin{aligned}
\mathbb{P}[\mathcal{E}_1] = \leq \mathbb{P}[\mathcal{E}_1|\widehat{\mathcal{H}}_{i,j} = \mathcal{H}_z] \; + \quad & \mathbb{P}[\mathcal{E}_1|\widehat{\mathcal{H}}_{i,j} = \mathcal{H}_s] \\
+ \quad & \mathbb{P}[\mathcal{E}_1|\widehat{\mathcal{H}}_{i,j} = \mathcal{H}_d \cup \mathcal{H}_m] \\
\leq e^{-\frac{N^{\mu-\alpha}(1-2\eta)^2}{8}} + 2e^{-\frac{N^{\mu-\alpha}(1-4\eta)^2}{16}} & \\
+ 2e^{-\frac{N^{\mu+\alpha-1}(1-6\eta)^2}{16}} + e^{-\frac{N^{\mu+\alpha-1}(1-6\eta)^2}{16}} & \\
\leq 6e^{-\frac{N^{\mu+\alpha-1}(1-6\eta)^2}{16}} &
\end{aligned}$$

where the inequalities in the third line are due to Lemmas A.1, A.2, A.3 and A.4 respectively provided in Appendix A. □

5.2 Position Identification

Lemma 5.2. *Given that a bin (i, j) is correctly classified as a singleton, the probability of error in identifying the position of the non-zero variable node can be upper bounded by*

$$\begin{aligned}
\mathbb{P}[\mathcal{E}_2] \leq & \exp\left\{-\frac{N^{\mu+\alpha-1}(1-2\eta)^2(c_1^2-1)}{8(c_1^2+1)}\right\} \\
+ \quad & \exp\left\{-\frac{N^{\mu+\alpha-1}\,(c_1(1-2\eta)-1)^2}{8(1+c_1^2)}\right\}
\end{aligned}$$

Proof. The detailed proof is provided in Appendix B. □

5.3 Peeling Process

d	2	3	4	5	6	7	8
δ	1.000	0.4073	0.3237	0.2850	0.2616	0.2456	0.2336
$d\delta$	2.000	1.2219	1.2948	1.4250	1.5696	1.7192	1.8688

Table 2: Constants for various error floor values

To analyze the peeling process alone independently, we refer to a *oracle based peeling decoder* which has the accurate classification of

all the bins and can accurately identify the position of the singleton given a singleton bin at any iteration. In other words, oracle based peeling decoder is the peeling part of our decoding scheme but with the assumption that the bin classification and position identification are carried out without any error.

Lemma 5.3 (Exact Matching). *For the exact matching case, choose $F^{d-1} = \delta N^{\alpha}$ where δ is chosen as given in Table. 2. Then the oracle based peeling decoder:*

- *successfully uncovers all the L matching positions if $L = \Omega(N^{\alpha})$ and $L \leq N^{\alpha}$, with probability at least $1 - O(1/N^{\frac{1}{d}})$*
- *successfully uncovers all the L matching positions, if $L = o(N^{\alpha})$, with probability at least $1 - e^{-\beta\varepsilon_1^2 N^{\alpha/(4l+1)}}$ for some constants $\beta, \varepsilon_1 > 0$ and $l > 0$.*

Proof. We borrow this result from Pawar and Ramchandran's [12]. Although our RSDIFT framework and their robust-FFAST scheme have three main differences:

- We are computing smaller IDFT's to recover a sparse bigger IDFT whereas in [12] the same is true for DFT instead of IDFT.
- Our problem model is such that the sparse components of the signal space has only positive amplitude and thus our bin processing part (bin classification and position identification) is different when compared to [12].
- The sparsity of the signal L to be recovered is exactly known in the case of [12] whereas we have no information about L not even the order with which the quantity scales in N.

Irrespective of these differences, the Tanner graph representation of the framework and the peeling part of the decoder are identical to that of the robust-FFAST scheme. And thus the limit of the *oracle based peeling decoder* for our scheme is identical to that in the robust-FFAST scheme [12]. With respect to the third difference, in robust-FFAST scheme the authors choose $F^{d-1} = \delta k$ where k is the sparsity of the signal (which is assumed to be known) and show the first assertion of the lemma. They also showed that upto a constant fraction $(1 - \varepsilon)$ of k-variables node can be recovered with probability of failure that decays exponentially in N. In our case, since $L = o(N^{\alpha})$, this result translates to recovering all the L non-zero variable nodes with an exponentially decaying failure probability. □

In any iteration, given a singleton bin, the peeling process, in the case of approximate matching, runs the Singleton-Decoder algorithm on the bin only if it was either originally a singleton or originally a double-ton with one of the variable nodes being peeled off already. This is in contrast to the exact matching case where the peeling decoder runs the Singleton-Decoder on the bin irrespective of it's original degree. Hence we need to analyze the oracle based peeling decoder for the approximate matching case separately compared to the exact matching case.

Lemma 5.4 (Approximate Matching). *For the approximate matching case, choose parameter $d \geq 8$ as described in Sec. 4.1.2 and $F^{d-1} = 0.7663N^{\alpha}$. Then the oracle based peeling decoder:*

- *successfully uncovers all but a small fraction $\varepsilon = 10^{-3}$ of the L matching positions, if $L = \Omega(N^{\alpha})$ and $L \leq N^{\alpha}$ with a failure probability that decays exponentially in N*
- *successfully uncovers all the L matching positions, if $L = o(N^{\alpha})$, with probability at least $1 - e^{-\beta\varepsilon_1^2 N^{\alpha/(4l+1)}}$ for some constants $\beta, \varepsilon_1 > 0$ and $l > 0$.*

Proof. As mentioned earlier, the key difference in the approximate matching case is peeling off variable nodes from only singleton and double-tons. An identical peeling decoder is used and analyzed in the problem of group testing [8] by Lee, Pedarsani and Ramchandran which the authors refer to as SAFFRON scheme. In SAFFRON, the authors claim that for a graph ensemble which has a regular degree of d on the variable nodes and a Poisson degree distribution on the bins, this peeling decoder with a left degree of $d = 8$ and a total number of bins at least equal to $6.13k$ recovers at least $(1 - \varepsilon)$ fraction of the k non-zero variable nodes with exponentially decaying probability. Note that $\frac{6.13}{8} \approx 0.7663$ is approximately the number of bins per stage for $d = 8$. We also leverage the result from [12] that the Tanner graph representation of the robust-FFAST (or equivaently RSDIFT framework) has a Poisson degree distribution on the bins. Combining these two results gives us the required results. □

Theorem 5.5 (Overall Probability of Error). *The RSIDFT framework succeeds with high probability asymptotically in N if the number of samples in each branch $f_i = N^{\alpha} + O(1)$ satisfies the condition $\alpha > 1 - \mu$.*

Proof. The overall probability of error $\mathbb{P}[\mathcal{E}_{\text{total}}]$ can be bounded using an union bound on the three error events $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$ given by

$$\mathbb{P}[\mathcal{E}_{\text{total}}] \;\leq\; \mathbb{P}[\mathcal{E}_1] + \mathbb{P}[\mathcal{E}_2] + \mathbb{P}[\mathcal{E}_3]$$

Using the expressions for error probabilities from Lemmas 5.1, 5.2, 5.3 and 5.4, we can see that all the terms vanish to zero as $N \to \infty$ if $\mu + \alpha - 1 \geq 0$, i.e. $\alpha \geq 1 - \mu$. □

6 SAMPLE AND COMPUTATIONAL COMPLEXITY

In this section, we will analyze the sketching complexity which is the number of samples we access from the sketch of the signal $\vec{x}$ stored in the database and the computational complexity as a function of the system parameters.

6.1 Sample Complexity

In each branch of the RSDIFT framework we down-sample the N samples by a factor of $\frac{N}{f_i}$ to get $f_i \approx N^{\alpha}$ samples. We repeat this for a random shift in each branch for $B = O(\log N)$ branches in each stage thus resulting in a total of $O(N^{\alpha} \log N)$ samples per block per stage. We repeat this for $d = \frac{1}{1-\alpha}$ such stages resulting in a total of $dN^{\alpha} \log N$ samples per block. So, the total number of samples is given by

$$S = O\left(dN^{\alpha} \log N\right) = O(N^{1-\mu} \log N)$$

6.2 Computational Complexity

As described in Eq. 3, the computation of $\vec{\mathrm{r}}$ involves three steps:

(1) Operation - I: Since we assume that the sketch of database $\vec{\mathrm{x}}$, $\mathcal{F}_N\{\vec{\mathrm{x}}\}$, is pre-computed, we do not include this in computational complexity.

(2) Operation - II: As described in Sec. 4.2, in each branch (i, j), we use a folding based technique to compute the sketch of $\vec{\mathrm{y}}$, $\mathcal{F}_N\{\vec{\mathrm{y}}'\}$ at points in the set $\mathcal{S}_{i,j}$. The folding technique involves two steps: folding and adding (aliasing) which has a complexity of $O(M)$ computations , and computing f_i-point DFTs that takes $O(N^\alpha \log N^\alpha)$ computations. So, for a total of dB branches the number of computations in this step is given by

$$\begin{aligned} C_{II} &= dB\,(\underbrace{N^\mu}_{\text{Folding}} + \underbrace{N^\alpha \log N^\alpha}_{\text{Shorter FFTs}}) \\ &= O(\max(N^{1-\mu}\log^2 N, N^\mu \log N)). \end{aligned}$$

Note: Folding and adding, for each shift, involves adding $N^{1-\alpha}$ vectors of length N^α. We know that the length of the query is $M = N^\mu$, i.e., the number of non-zero elements in $\vec{\mathrm{y}}$ (zero-padded version of the query) is M and hence we only need to compute M additions instead of length of the vector N.

(3) Operation - III: Computing $\mathcal{F}_N^{-1}\{\vec{\mathrm{x}}\}$ involves two parts: RSIFT framework and the decoder. The RSIFT framework involves computing smaller f_i point IDFTs, which takes approximately $O(N^\alpha \log N^\alpha)$ computations in each branch. For a total of dB branches, we get a complexity of $O(dBN^\alpha \log N^\alpha)$. In the decoding process, the dominant computation is from position identification. Each position identification process involves correlating the observation vector of length B with $\frac{N}{f_i} \approx N^{1-\alpha}$ column vectors, which amounts to $BN^{1-\alpha}$ computations. There will be a maximum of dL such position identifications, which gives a complexity of$O(dLBN^{1-\alpha})$. Now, plugging in $\alpha = 1-\mu$ (condition for vanishing probability of error) the total number of computations involved in this step, C_I, is given by

$$\begin{aligned} C_{III} &= dB\,(\underbrace{O(N^\alpha \log N^\alpha)}_{\text{Shorter IFFTs /block/stage}} + \underbrace{L\,N^{1-\alpha}}_{\text{Correlations}}) \\ &= O(\max(N^{1-\mu}\log^2 N, N^{\mu+\lambda}\log N)). \end{aligned}$$

Thus, the total number of computations, $C = \max\{C_{II}, C_{III}\}$, is given by

$$C = O(\max\{N^{1-\mu}\log^2 N, N^{\mu+\lambda}\log N\})$$

7 SIMULATION RESULTS

Simulations were carried out to test the performance of RSIDFT framework for exact matching scenario on a database of length $N = 10^{12}$ for two different query lengths $M = 10^5$ ($\mu = 0.41$) and $M = 10^3$ ($\mu = 0.25$). The database was generated as a equiprobable $\{+1, -1\}$ sequence of length N. A substring of length M from the generated database is presented as a query. Also the chosen query was repeated at $L = 10^6$ randomly chosen locations in the database.

The sample gain, defined as the ratio of N to the number of samples used from the sketch of database, was varied and the probability of RSIDFT framework to miss a match (P_e), as defined below, was measured.

$$P_e = \frac{\#\text{ of correctly identified locations}}{L}$$

The plots of P_e vs. sample gain, is presented in Fig. 4 for two different query lengths: $M = 10^5$ ($\mu = 0.41$) in Fig 4(a) and $M = 10^3$ ($\mu = 0.25$) in Fig 4(b). As can be inferred from the plots we achieve a sample gain of 200-500 (depending on the tolerable error probability) for the query length corresponding to $\mu = 0.41$ and a sample gain of 2-8 for $\mu = 0.25$. This sample gain results from an average number of samples per branch $f_i \approx 9.25\times10^7$ ($\alpha = 0.66$) for $\mu = 0.41$, and $f_i \approx 6.94 \times 10^9$ ($\alpha = 0.82$) for $\mu = 0.25$. The trend in the results almost matches with the theoretical findings of $\alpha = 1-\mu$. We also notice a sharp threshold in the sample gain, below which the RSIDFT framework succeeds with very high probability.

A BIN CLASSIFICATION ERRORS

We employ classification rules based only on the first element of the measurement vector at bin (i, j) which can be given by

$$Z[1] = \begin{cases} \sum_{\ell=0}^{g_i-1}\sum_{k=0}^{M-1} n_{l,k} & \text{if } \mathcal{H} = \mathcal{H}_z \\ M_1 + \sum_{\ell=0}^{g_i-2}\sum_{k=0}^{M-1} n_{l,k} & \text{if } \mathcal{H} = \mathcal{H}_s \\ M_1 + M_2 + \sum_{\ell=0}^{g_i-3}\sum_{k=0}^{M-1} n_{l,k} & \text{if } \mathcal{H} = \mathcal{H}_d \end{cases} \tag{9}$$

where $n_{l,k} = x[\theta_\ell + k]y[k]$ and $\theta_\ell \notin \{\tau_1, \tau_2, \ldots, \tau_L\}$. Also for the case of exact matching $M_1 = M_2 = M$ whereas in the case of approximate matching the values of $M_1, M_2 \in [M(1-2\eta) : M]$.

Lemma A.1 (zero-ton). *Given that the bin (i, j) is a zero-ton, the classification error can be bounded by*

$$\mathbb{P}[\mathcal{E}_1|\mathcal{H}_z] \le e^{-\frac{N^{\mu+\alpha-1}(1-2\eta)^2}{8}}$$

Lemma A.2 (singleton). *Given that the bin (i, j) is a singleton, the classification error can be bounded by*

$$\mathbb{P}[\mathcal{E}_1|\mathcal{H}_s] \le 2e^{-\frac{N^{\mu+\alpha-1}(1-4\eta)^2}{16}}$$

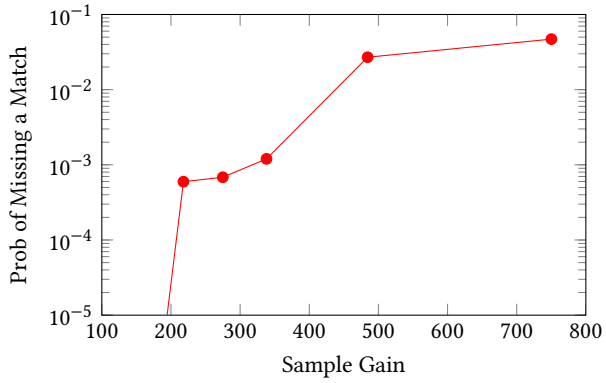
Lemma A.3 (double-ton). *Given that the bin (i, j) is a double-ton, the classification error can be bounded by*

$$\mathbb{P}[\mathcal{E}_1|\mathcal{H}_d] \le 2e^{-\frac{N^{\mu+\alpha-1}(1-6\eta)^2}{16}}$$

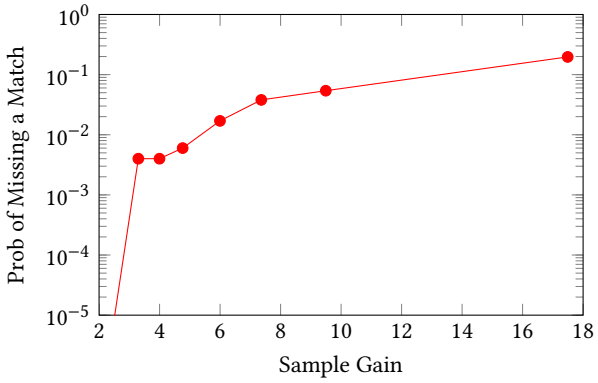
Lemma A.4 (multi-ton). *Given that the bin (i, j) is a multi-ton, the classification error can be bounded by*

$$\mathbb{P}[\mathcal{E}_1|\mathcal{H}_m] \le e^{-\frac{N^{\mu+\alpha-1}(1-6\eta)^2}{16}}$$

Proof. The proof for Lemmas A.1 A.2, A.3, A.4 follows similar lines as in Lemma A.1. For a detailed proof refer to our longer version [7] (Lemmas 9,10,11). □



(a) $M = 10^5 (\mu = 0.41)$, $\tilde{N} = 10^7$, $G = 10^5$



(b) $M = 10^3 (\mu = 0.25)$, $\tilde{N} = 10^6$, $G = 10^6$

Figure 4: Plots of probability of missing a match vs. sample gain for exact matching of a query of length M from a equiprobable binary {+1,-1} sequence of length $N = 10^{12}$, divided into G blocks each of length $\tilde{N}$. The substring was simulated to repeat in $L = 10^6 (\lambda = 0.5)$ locations uniformly at random.

B POSITION IDENTIFICATION

We will analyze the singleton identification in two separate cases:

- $\mathcal{E}_{21}$: Event where the position is identified incorrectly when the bin is classified correctly a singleton
- $\mathcal{E}_{22}$: In the case of approximate matching, event where the position is identified incorrectly when the bin is originally a double-ton and one of the non-zero variable nodes has already been peeled off

Definition B.1 (Mutual Incoherence). The mutual incoherence $\mu_{\max}(\mathbf{W})$ of a matrix $\mathbf{W} = [\vec{w}_1\ \vec{w}_2\ \cdots \vec{w}_i \cdots \vec{w}_N]$ is defined as

$$\mu_{\max}(\mathbf{W}) \triangleq \max_{\forall i \neq j} \frac{|\vec{w}_i^{\dagger} \vec{w}_j|}{||\vec{w}_i||.||\vec{w}_j||}$$

Lemma B.2 (Mutual Incoherence Bound for sub-sampled IDFT matrix [[12],Proposition A.1). *The mutual incoherence μ_{max} $(\mathbf{W_{i,k}})$ of the sensing matrix $\mathbf{W}_{i,k}$ (defined in Eq. 6), with B shifts, is upper bounded by*

$$\mu_{max} < 2\sqrt{\frac{\log(5N)}{B}}$$

Proof. The proof follows similar lines as the proof for Lemma V.3. in [12]. □

Lemma B.3. *For some constant $c_1 \in \mathbb{R}$ and the choice of $B = 4c_1^2 \log 5N$, the probability of error in identifying the position of a singleton at any bin (i, j) can be upper bounded by*

$$\mathbb{P}[\mathcal{E}_{21}] \leq \exp\left\{-\frac{N^{\mu+\alpha-1}(1-2\eta)^2(c_1^2-1)}{8(c_1^2+1)}\right\}$$

Lemma B.4. *For some constant $c_1 \in \mathbb{R}$ and the choice of $B = 4c_1^2 \log 5N$, the probability of error in identifying the position of second non-zero variable node at a double-ton at any bin (i, j), given that the first position identification is correct, can be upper bounded by*

$$\mathbb{P}[\mathcal{E}_{22}] \leq \exp\left\{-\frac{N^{\mu+\alpha-1}\ (c_1(1-2\eta)-1)^2}{8(1+c_1^2)}\right\}$$

Proof. The proof for Lemmas B.3 and B.4 can be found in our longer version[7] (Lemma 14,15). □

REFERENCES

[1] Amihood Amir, Moshe Lewenstein, and Ely Porat. 2004. Faster algorithms for string matching with k mismatches. *Journal of Algorithms* 50, 2 (2004), 257–275.

[2] Alexandr Andoni, Haitham Hassanieh, Piotr Indyk, and Dina Katabi. 2013. Shift finding in sub-linear time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 457–465.

[3] Robert S Boyer and J Strother Moore. 1977. A fast string searching algorithm. *Commun. ACM* 20, 10 (1977), 762–772.

[4] William I Chang and Thomas G Marr. 1994. Approximate string matching and local similarity. In *Annual Symposium on Combinatorial Pattern Matching*. Springer, 259–273.

[5] Paolo Ferragina and Giovanni Manzini. 2005. Indexing compressed text. *Journal of the ACM (JACM)* 52, 4 (2005), 552–581.

[6] Haitham Hassanieh, Fadel Adib, Dina Katabi, and Piotr Indyk. 2012. Faster GPS via the sparse Fourier transform. In *mobicom*. ACM, 353–364.

[7] Nagaraj T Janakiraman, Avinash Vem, Krishna R Narayanan, and Jean-Francois Chamberland. 2017. Sub-string/Pattern Matching in Sub-linear Time Using a Sparse Fourier Transform Approach. *arXiv preprint arXiv:1704.07852* (2017).

[8] Kangwook Lee, Ramtin Pedarsani, and Kannan Ramchandran. 2015. SAFFRON: A fast, efficient, and robust framework for group testing based on sparse-graph codes. *arXiv preprint arXiv:1508.04485* (2015).

[9] Heng Li and Richard Durbin. 2009. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics* 25, 14 (2009), 1754–1760.

[10] Heng Li and Richard Durbin. 2010. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics* 26, 5 (2010), 589–595.

[11] Gonzalo Navarro. 2001. A guided tour to approximate string matching. *ACM computing surveys (CSUR)* 33, 1 (2001), 31–88.

[12] Sameer Pawar and Kannan Ramchandran. 2014. A robust R-FFAST framework for computing a k-sparse n-length DFT in O(k log n) sample complexity using sparse-graph codes. In *Proc. IEEE Int. Symp. Information Theory*. 1852–1856.

[13] Nan Zhang, Amar Mukherjee, Don Adjeroh, and Tim Bell. 2003. Approximate Pattern Matching Using the Burrows-Wheeler Transform. In *Proceedings of the Conference on Data Compression*. IEEE Computer Society, 458.