Coding for IBLTs with Listing Guarantees

Daniella Bar-Lev, **Avi Mizrahi**, **Tuvi Etzion**, **Ori Rottenstreich**, and **Eitan Yaakobi**
Department of Computer Science, Technion—Israel Institute of Technology, Haifa 3200003, Israel
Email: {daniellalev, avraham.m, etzion, or, yaakobi}@cs.technion.ac.il

***Abstract*—The *Invertible Bloom Lookup Table* (*IBLT*) is a probabilistic data structure for set representation, with applications in network and traffic monitoring. It is known for its ability to list its elements, an operation that succeeds with high probability for sufficiently large table. However, listing can fail even for relatively small sets. This paper extends recent work on the worst-case analysis of IBLT, which guarantees successful listing for all sets of a certain size, by introducing more general IBLT schemes. These schemes allow for greater freedom in the implementation of the insert, delete, and listing operations and demonstrate that the IBLT memory can be reduced while still maintaining successful listing guarantees. The paper also explores the time-memory trade-off of these schemes, some of which are based on linear codes and B_h-sequences over finite fields.**

I. Introduction

The *Invertible Bloom Lookup Table* (*IBLT*) is a probabilistic data structure that is used for representing dynamic sets, with the ability to list the elements in the set [3], [5]. It has several applications, including traffic monitoring, error-correction codes for large data sets, and set reconciliation between two or more parties [4], [6]–[10], [13], [16]. Although the listing operation of an IBLT might fail, it has been shown to be highly successful when the allocated memory is proportional to the number of elements [5]. However, it is still possible to encounter failure in certain instances, such as when several elements are mapped to the same k entries.

In this work, we consider the case of IBLT with listing guarantees in the worst-case. Our point of departure here is a recent work [11] which introduced the problem of designing an IBLT with listing guarantees. Under this setup, listing elements always succeeds for any set of up to d elements from a finite universe. They describe the mapping of elements to cells of the IBLT using a binary matrix, where each column represents an element from the universe and each row represents a cell of the IBLT. Such a matrix is called d-*decodable* if the listing is guaranteed to be successful for any set of up to d elements.

In [11], the authors restricted the problem definition to only consider the so-called d-*decodable matrices* and assumed that the element insertion, deletion, and listing operations as well as the structure of the IBLT were the same as the traditional IBLT (Fig. 1). This work extends the definition of a d-decodable matrix to a d-*decodable scheme* by allowing greater freedom in the implementation of the insert, delete and listing operations and the design of the IBLT cells. Such an IBLT scheme, which is composed of a table, mapping matrix, and a set of operations, is called d-*decodable* if its listing operation on its table is guaranteed to be successful whenever the number of elements in it is at most d. We show that such schemes can reduce the memory size of the IBLT, while still maintaining successful listing guarantees. The paper also explores the time-memory trade-off of these schemes, some of which are based on linear codes and B_h-sequences over finite fields.

This paper is organized as follows. Section II introduces the definitions that are used throughout the paper and the problem statements. Section III presents our constructions and lower bounds. Lastly, in Section IV we discuss the time-memory trade-off for the different constructions. Due to space limitations, some of the proofs will appear in the extended version of the paper.

D. Bar-Lev and T. Etzion were supported in part by ISF grant no. 222/19. A. Mizrahi and O. Rottenstreich were partially supported by Israel PBC-VATAT and by the Technion Center for Machine Learning and Intelligent Systems (MLIS). E. Yaakobi and D. Bar-Lev were funded by the European Union (ERC, DNAStorage, 865630). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

II. Definitions and Problem Statement

We start by formally defining IBLT schemes and their variants.

Definition 1. An **IBLT *scheme*** consists of the following:

1) A finite universe U_n of size n of all possible elements.
2) A *lookup table* T which is a data structure that is composed of m cells each of size b bits. The size of the table T is denoted by $s(T) = mb$.
3) An **IBLT *protocol*** is a set of algorithms $P = (I, D, M, L)$ which are defined as follows:
   - 3.1. *Insert algorithm* $I : U_n \times [m] \times \{0,1\}^b \to \{0,1\}^b$, which receives an element $u \in U_n$ (assuming it is not stored in the IBLT) and a cell state $(i, T_i) \in [m] \times \{0,1\}^b$ that contains the index and the content of the cell T_i. This algorithm updates T_i to capture the insertion of u into T.
   - 3.2. *Delete algorithm* $D : U_n \times [m] \times \{0,1\}^b \to \{0,1\}^b$, which receives an element $u \in U_n$ (assuming it is stored in the IBLT) and a cell state $(i, T_i) \in [m] \times \{0,1\}^b$ that contains the index i and the content of the cell T_i. This algorithm updates T_i to capture the deletion of u from T.
   - 3.3. *Mapping algorithm* $M : U_n \to 2^{[m]}$ which maps an element $u \in U_n$ to a subset of the cells in the lookup table (i.e., all cells that should be modified by the insertion/deletion of the element u).
   - 3.4. *Listing algorithm* $L : \{0,1\}^{mb} \to 2^{U_n}$ which either lists all the elements that are stored in the IBLT, or fails.

An IBLT scheme (U_n, T, P) is called d-***decodable*** if it satisfies the ***successful listing*** property: if the lookup table T stores a set $S \subseteq U_n$ of at most d elements, then $L(T) = S$.

The definition of the IBLT protocol generalizes the well known definition of the traditional IBLT [3], [5]. In our

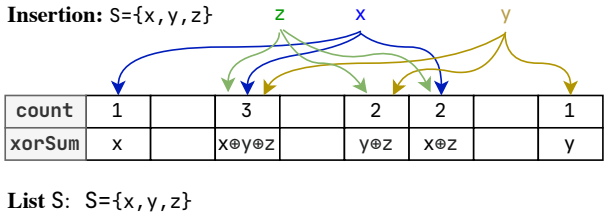

Figure 1. An IBLT representing the set $S = \{x, y, z\}$. Here the table T_s consists of $m = 8$ cells, each composed of a count field and a xorSum field. Element listing is possible starting from the most left cell for which the count field equals one, identifying x as a member of S and removing it from the other two cells. Then both y and z can be listed by the new pure cells.

terminology, the traditional IBLT scheme, which we refer wherein as the standard scheme, can be described as follows.

Definition 2. An IBLT scheme is called ***standard***, and is denoted by $(U_n, T_s, P_s = (I_s, D_s, M, L_{\text{peeling}}))_S$, if the followings three conditions hold. (1) The table T_s is composed of m cells, typically each of size $b = 2\log n$. Each cell is composed of two components, a counter field, and a data field, which is referred as the *xorSum field*. (2) For $u \in U_n$, cell index $i \in [m]$, and cell state $T_i = (t_{\text{count}}, t_{\text{xorSum}}) \in \{0,1\}^{\log n} \times \{0,1\}^{\log n}$ we have that $I_s(u, i, T_i) = (t_{\text{count}} + 1, t_{\text{xorSum}} \oplus u)$ and $D_s(u, i, T_i) = (t_{\text{count}} - 1, t_{\text{xorSum}} \oplus u)$. (3) L_{peeling} is a listing algorithm that operates as follows. First, it looks for a *pure cell*, which is a cell whose counter is 1, and deletes the corresponding element from the lookup table T_s using the delete algorithm D_s. It continues this way until all elements are extracted from the lookup table or no pure cell exists. In the latter case, we say that the listing algorithm *fails*.

In most works which studied IBLTs, the mapping M in the standard scheme is implemented using a set of hash functions, each maps an element of U_n to a subset of $[m]$. Here, $M(u)$ for $u \in U_n$ is defined to be the outputs of these functions. An example of a standard IBLT scheme is depicted in Fig. 1.

Definition 3. An IBLT scheme is called ***standard-indel***, denoted by $(U_n,T,P)_{SI}$, if its insert and delete algorithms are the same as in the standard schemes, i.e., $I{=}I_s$ and $D{=}D_s$.[1]

Hence, the main difference between a standard scheme and a standard-indel scheme is that while the former requires L_{peeling} to be the listing algorithm, this algorithm in the latter can be arbitrary. Furthermore, note that by definition, a standard IBLT scheme implies a standard-indel IBLT scheme, which implies an IBLT scheme. In the cases where the IBLT scheme is a standard scheme or a standard indel scheme, we describe the mapping using an $m \times n$ matrix M, in which $M_{i,j} = 1$ if and only if the j-th element of U_n is mapped to the i-th cell of T.

Example 1. For the universe $U_6 = \{1, \ldots, 6\}$ consider the following binary matrix M.

$$M = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

[1]This implies that every cell consists of a xorSum field and a counter field. However, we relax the assumption that the counter field is of size $\log n$. If necessary, the summation operations are calculated modulo the maximum number the counter can represent.

An IBLT based on M has $m = 5$ cells, each associated with a row of M. Such an IBLT, when containing for instance the set $S = \{1, 3, 4\} \subseteq [6]$, has the counter array $(2, 1, 2, 0, 1)$ as the sum of the entries in the first, third, and fourth columns.

In this work we explore d-decodable IBLT schemes. Naturally, we seek to study the effect of the universe size n, the decodable threshold value d, and the protocol P on the memory size used by the lookup table, $s(T)$. Additionally, we aim to find IBLT protocols that minimize the memory size $s(T)$. This problem can be formalized as follows.

Problem 1. Given a set U_n, an integer $d \leq n$, find the values
1) $s^*_S(n,d) = \min\{s(T) : (U_n, T, P)_S \text{ is } d\text{-decodable}\}$,
2) $s^*_{SI}(n,d) = \min\{s(T) : (U_n, T, P)_{SI} \text{ is } d\text{-decodable}\}$,
3) $s^*(n,d) = \min\{s(T) : (U_n, T, P) \text{ is } d\text{-decodable}\}$.

To limit the number of read or write memory accesses to a small fixed number when querying or inserting elements, we consider a specific family of IBLT schemes which we denote by (d,k)-decodable schemes. A d-decodable IBLT scheme is called (d,k)-***decodable*** if for any $u \in U_n$, its insert and delete algorithms affect exactly k cells of T. This definition is extended also to the standard and standard-indel schemes. Similarly to Problem 1, we are interested in (d,k)-decodable IBLT schemes in which the size of the lookup table is minimal.

Problem 2. Given a set U_n, integers $d, k \leq n$, find the values
1) $s^*_S(n,d,k) = \min\{s(T) : (U_n, T, P)_S \text{ is } (d,k)\text{-decodable}\}$,
2) $s^*_{SI}(n,d,k) = \min\{s(T) : (U_n, T, P)_{SI} \text{ is } (d,k)\text{-decodable}\}$,
3) $s^*(n,d,k) = \min\{s(T) : (U_n, T, P) \text{ is } (d,k)\text{-decodable}\}$.

We note that the results in this work demonstrate that the size of the lookup table T can be reduced significantly by using general IBLT schemes as compared to the standard and standard-indel schemes. It is noteworthy to mention that this reduction in size may come with a trade-off in terms of increased computational time for the algorithms. A more detailed discussion of this trade-off is presented in Section IV. In the rest of this work, we assume that b is at most $2\log(n)$.

III. Decodable Schemes

So far in the literature only standard IBLT schemes were studied, while the other two variants are new to this work.

A. Standard Schemes

In this section we review previous results on standard decodable IBLT schemes, which are obtained by only selecting the mapping algorithm of the standard protocol. Each scheme is defined by an $m \times n$ binary mapping matrix M, and the size of its table is given by $s(T_s) = 2m\log n$. Table I summarizes the relevant results from [11, Table 1]. These are originally given as bounds on m^*, the minimal number of rows in a matrix M, which implies bounds on s^*_S. For more results and details the reader is referred to the original work [11].

B. Standard-Indel Schemes

This section considers the case of standard-indel schemes, i.e., schemes in which the insertion and deletion algorithms are as in the standard schemes. First, we note that to guarantee successful listing of up to d elements, it is sufficient to use counters of size $\log d$, with modulo d arithmetic.

Claim 1. It holds that $s^*_{SI}(n,d) \leq \frac{\log(n)+\log(d)}{2\log(n)} \cdot s^*_S(n,d)$.

Our main result in this section is that the memory size, $s(T)$, can be strictly reduced compared to the standard schemes by

Table I. **Constructions and lower bounds overview**. Note that $\Psi(n,d)$ is the maximum number of elements in a B_d-sequence in $GF(n)$ and $\lambda(n,k)$ is the minimal integer m such that the number of unique length-m binary vectors with weight k is at least n.

Scheme	d	k	s^* - lower bound	s^* - upper bound	b	Theorem
standard [11]	3	–		$2\left\lceil \frac{3}{\log 3}\log n\right\rceil \lceil \log n\rceil$	$2\lceil \log n\rceil$	[11]
	$O(1)$	–		$O\left(\log_2^{\lfloor \log_2 d\rfloor+1} n\right)$	$2\lceil \log n\rceil$	[11]
	3	2	$2\lceil 2\sqrt{n}\rceil \lceil \log n\rceil$	$2\lceil 2\sqrt{n}\rceil \lceil \log n\rceil$	$2\lceil \log n\rceil$	[11]
	5	3		$6\left(2\left\lceil \sqrt{n/6}\right\rceil+1\right)\lceil \log n\rceil$	$2\lceil \log n\rceil$	[11]
	7	4		$8\left\lceil \sqrt{2n}\right\rceil \lceil \log n\rceil$	$2\lceil \log n\rceil$	[11]
	4	$2\lceil \log(n+1)\rceil$		$8\lceil \log(n+1)\rceil \lceil \log n\rceil$	$2\lceil \log n\rceil$	[11]
	any	d		$2d\lceil \sqrt{n}\rceil \lceil \log n\rceil$	$2\lceil \log n\rceil$	[11]
	any	$O(d\log n)$		$O\left(d^2\cdot \log^2 n\right)$	$2\lceil \log n\rceil$	[11]
standard-indel (this work)	3	–		$(\log n+2)(\log n+1)$	$\log n+2$	Theorem 1
	4	–		$\big(\log(n+1)+2\big)\big(2\log(n+1)+1\big)$	$\log(n+1)+2$	Theorem 1
	3	–		$(\log n+1)^2$	$\log n+1$	Theorem 2
	3	any		$(\log n+1)\big(\lambda(n,k)+1\big)$	$\log n+1$	Theorem 2
general (this work)	any	–	$d\big(\log(n)-\log(d)\big)$	$d\log(n+1)$	$\log(n+1)$	Lemma 1,Theorem 3
	any	1	$\left\lceil \frac{n}{\Psi(n,d)-1}\right\rceil \log(n)$	$\left\lceil 2n/\big(\sqrt[d]{n}-2\big)\right\rceil \log(n)$	$\log(n)$	Theorem 5,Corollary 3
	$d>2$, even	2		$\left(\left\lceil 2n/\big(\sqrt[d]{n^2}-2\big)\right\rceil+1\right)\log(n)$	$\log(n)$	Theorem 6
	4	2		$\left(2\left\lceil 2n/(3\sqrt{n}-4)\right\rceil+1\right)\log(n)$	$\log(n)$	Theorem 7

utilizing improved versions of the listing algorithm rather than L_{peeling}. As stated in Definition 2, the stopping condition of the peeling process is the absence of pure cells, i.e., there are non-empty cells, but none of them have a counter of 1. Next, we show that by adding a small step to this process, we can guarantee successful listing for small values of d. The key idea of this algorithm, denoted by L'_{peeling}, is to overcome the absence of pure cells by searching for two cells which can reveal together the next element in the table. This is formally defined in Algorithm 1, where the values of $t_{\text{count}}, t_{\text{xorSum}}$ in the i-th cell of T is denoted by $T^i_{\text{count}}, T^i_{\text{xorSum}}$, respectively.

Algorithm 1: Extended peeling algorithm L'_{peeling}

Input: $d, T=(t_1,\ldots,t_m)$
Output: $S\subseteq U$, the elements that are stored in T

$S\leftarrow\emptyset$
while T *has a non-empty cell* **do**
  $J\leftarrow L_{\text{peeling}}(T)$
  if $J=\emptyset$ **then**
    if *there exist* T_i,T_j *such that* $T^i_{count}=\max_\ell\{T^\ell_{count}\}$ *and* $T^i_{count}-T^j_{count}=1$ **then**
      $e\leftarrow T^i_{\text{xorSum}}\oplus T^j_{\text{xorSum}}$
      Delete e from T
      $J\leftarrow J\cup\{e\}$
    else return *Failure* ;
  end
  $S\leftarrow S\cup J$
end
return S

The advantages of the standard-indel schemes and L'_{peeling} over the standard schemes are presented next.

Theorem 1. The following holds.
1) $s^*_{SI}(n,3)\le(\log n+2)(\log n+1)$ for $n=2^r$,
2) $s^*_{SI}(n,4)\le\big(\log(n+1)+2\big)\big(2\log(n+1)+1\big)$ for $n=2^r-1$.

Proof: We start by proving the second bound. Let M' be a binary parity check matrix for a length-n linear code with Hamming distance 5, e.g. a parity check matrix for the binary Bose–Chaudhuri–Hocquenghem (BCH) code with $2\log(n+1)$ rows and n columns [14]. Let M be the matrix that is obtained from M' by adding the all-ones row as the last row. Let T be the table with $m=2\log(n+1)+1$ cells, each of size $b=\log(n+1)+2$. For each $i\in[m]$, we let $T_i=(t_{\text{count}},t_{\text{xorSum}})\in\{0,1\}^2\times\{0,1\}^{\log(n+1)}$ (i.e., the count field consists from two bits and the operations are done modulo 4). Hence $s(T)=\big(\log(n+1)+2\big)\big(2\log(n+1)+1\big)$. To prove the claim, we show that for the protocol $P=\left(I_s,D_s,M,L'_{\text{peeling}}\right)$ we have that $(U_n,T,P)_{SI}$ is a 4-decodable standard-indel scheme. Let $S\subseteq U_n$ be the set of elements in T, and assume $|S|\le 4$.

Note that for a code of distance d, any set C of at most $d-1$ columns of a parity check matrix is independent. Let C denote the set of columns in M which corresponds with the elements of S, and observe that since $|C|\le 4$ we have that the columns of C are independent, and their sum must contain an entry with an odd value. We show that L'_{peeling} always terminate successfully with the correct set S. At any step in the loop where L_{peeling} fails (line 5), there is no pure cell, which implies that $|S|\ge 3$.

If $|S|=3$, and since there is no pure cell, we have a cell T_i for which $t_{\text{count}}=2$ and $t_{\text{xorSum}}=u\oplus v$ for $u,v\in S$. Recall that the case where $|S|=3$ can be identified by $T^m_{\text{count}}=3$, and $T^m_{\text{xorSum}}=u\oplus v\oplus w$ (where $S=\{u,v,w\}$). Thus, $w=t_{\text{xorSum}}\oplus T^m_{\text{xorSum}}$ can be identified and deleted from T and the algorithm can continue. Otherwise, $|S|=4$, which can be identified since $T^m_{\text{count}}=0$ and T is not all zeroes. As mentioned, the sum of the 4 columns in C must contain an odd value entry, and since there is no pure cell, the latter implies that there exist a cell T_i with $t_{\text{count}}=3$. By arguments similar to the ones for $|S|=3$, one element from S can be correctly identified and removed and the algorithm can continue.

To prove the first bound, one can use similar arguments with the difference of letting M' be the binary matrix that is composed of all the different n columns of length $\log n$. ■

Let $\lambda(n,k)\triangleq\min\left\{m:\ n\ge\binom{m}{k}\right\}$ be the minimal integer m such that the number of unique length-m binary vectors with weight k is at least n. We can further improve the case where $d=3$ by cutting the cell counters width to one bit.

Theorem 2. For $n = 2^r$, we have that $s^*_{SI}(n,3) \leq (\log n+1)^2$, and $s^*_{SI}(n,3,k) \leq (\log n+1)\left(\lambda(n,k)+1\right)$, for any $k \geq 1$.

The proof of Theorem 2 is similar to Theorem 1, with the key idea that we can still identify the number of elements in the cells by utilizing the fields of the last cell.

C. General Schemes

In this section we consider general IBLT schemes (i.e., not standard or standard indel schemes). That is, here we allow the modification of the insert and delete algorithms as well as the listing and mapping algorithms and the design of the lookup table T. More precisely, we allow the insert and delete algorithms to rely on computations over finite fields. For an integer $r \geq 1$, let $GF(2^r)$ be the Galois Field of size 2^r. In the rest of this section all the operations are the field operations.

Define T_F to be the lookup table that consists of m cells, each of size r bits. For an $m \times n$ matrix $H \in GF(2^r)^{m\times n}$, let M_H be the mapping algorithm that maps the j-th element of U_n to the i-th cell of T_F if and only if $H_{i,j} \neq 0$. Additionally, let I_H and D_H be the insert and delete algorithms which are defined as follows. For $0 \leq i \leq m$ and $u_j \in U_n$, $j \in [n]$, let (i, T_i) be the current cell state of the i-th cell of T. Then

$$I_H(u_j, i, T_i) \triangleq T_i + H_{i,j}, \quad \text{and} \quad D_H(u_j, i, T_i) \triangleq T_i + H_{i,j}.$$

Note that if we consider T_F as a vector of length m over $GF(2^r)$, then the state of T_F after inserting or deleting an element $u_j \in U_n$ is equal to $T_F + h_j$, where h_j is the j-th column of H. Hence, since the characteristics of the field is 2, if the lookup table stores a set $S \subseteq U_n$ then the state of T_F is given by $T_F(S) = H \cdot v_S$, where $v_S \in \{0,1\}^n$ is the indicator binary length-n vector such that $\mathsf{supp}(v_S) = S$ (i.e., the j-th entry of v_S is one if and only if $j \in S$).

We start by showing that the size of the required lookup table T_F can be drastically reduced using I_H, D_H by selecting a suitable matrix H. To this end, first consider the following $d \times n$ matrix, for $n = 2^r - 1$, where α is primitive in $GF(2^r)$.

$$H^d_{n,\alpha} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & \alpha^{3\cdot 2} & \alpha^{3\cdot 3} & \cdots & \alpha^{3(n-1)} \\ 1 & \alpha^5 & \alpha^{5\cdot 2} & \alpha^{5\cdot 3} & \cdots & \alpha^{5(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2d-1} & \alpha^{(2d-1)2} & \alpha^{(2d-1)3} & \cdots & \alpha^{(2d-1)(n-1)} \end{bmatrix}$$

It is well known that $H^d_{n,\alpha}$ is a parity-check matrix of a primitive narrow-sense BCH code [14] with minimum distance $t \geq 2d+1$. That is, $H^d_{n,\alpha}$ is a parity-check matrix of a d-error-correcting-code of length $n = 2^r - 1$.

Theorem 3. For $n = 2^r - 1$ and $d \geq 1$, it holds that $s^*(n,d) \leq d \cdot r = d\log(n+1)$.

Proof: To prove the result, we show that for $H = H^d_{n,\alpha}$, there exists a listing algorithm L such that the scheme $(U_n, T_F, P = (I_H, D_H, M_H, L))$ is a d-decodable IBLT scheme. Since $s(T_F) = d\log(n+1)$ the result follows.

Note that H is a parity-check matrix of a code that corrects d errors. Hence, for any $S \subseteq U_n$ such that $|S| \leq d$, the syndromes $H \cdot v_S$ are unique. That is, for any $|S| \leq d$, the state of T_F is unique which implies that there exists a listing algorithm L for which $(U_n, T_F, P = (I_H, D_H, M_H, L))$ is d-decodable. ∎

The listing algorithm for the protocol in the latter proof can be implemented using a decoder for the BCH code. Some of the known decoders use the Berlekamp-Massey algorithm, which is very efficient. In [15], the authors presented an efficient decoder for large values of n with $\max\left\{O(d\sqrt{n}), O(d^2\log(n)\right\}$ time complexity. We note that the rest of this section considers mappings that are based on the parity-check matrix of the BCH code, and that the BCH decoders can be used for listing in these cases as well by introducing some adjustments with low time overhead.

Before we consider the case of (d,k)-decodable IBLT schemes, we present a lower bound on $s^*(n,d)$ in the following lemma.

Lemma 1. It holds that $s^*(n,d) > d\log(n) - d\log(d)$.

Corollary 1. If $d = O(\mathsf{polylog}(n))$ then $\lim_{n\to\infty} \frac{s^*(n,d)}{d\log(n)} = 1$.

Next we discuss IBLT schemes which are (d,k)-decodable. Clearly, for $k = d$ the d-decodable scheme that is presented in the proof of Theorem 3 is a (d,k)-decodable scheme which implies the following result.

Corollary 2. For $n = 2^r - 1$ and $d \geq 1$, we have that $s^*(n,d,k \geq d,P) \leq k\log(n+1)$. [2]

To discuss the more involved case of $k < d$, we first give the definition of B_h-sequences [2, Section V], [1], [12].

Definition 4. A sequence $g_1, \ldots, g_n$ in an Abelian group G is called a B_h***-sequence*** if all the non-zero sums $g_{i_1} + g_{i_2} + \cdots + g_{i_h}$, for $1 \leq i_1 \leq i_2 \leq \cdots \leq i_h \leq n$ are distinct in G.

It can be readily verified that any B_h-sequence is also a $B_{h'}$-sequence for $1 \leq h' \leq h$. The following is an example for a B_2-sequence.

Example 2. Let $G = (\mathbb{Z}, +)$ and consider the sequence $1, 2, 5, 7$. It can be verified that all the sums $a + b$ for $a, b \in \{1, 2, 5, 7\}$ are distinct and hence $1, 2, 5, 7$ is a B_2-sequence. Since we have that $1 + 1 + 7 = 2 + 2 + 5 = 9$, the latter sequence is not a B_3-sequence.

In this work, we only consider B_h-sequences in $GF(2^r)$. Denote by $\Psi(2^r, h)$ the maximum number of elements in such a B_h-sequence. In the next theorem, we present an upper bound on $s^*(n,d,k=1)$ under the assumption that the size of each cell in a lookup table T_F is $b = \log(n)$.

Theorem 4. Let $n = 2^r$. If the size of each cell in T is $b = \log(n)$, then $s^*(n,d,k=1) \leq \left\lceil \frac{n}{\Psi(n,d)-1} \right\rceil \log(n)$.

Proof: Let $\ell \triangleq \Psi(n,d) - 1$ and let $0, g_1, \ldots, g_\ell$ be a B_d-sequence of length $\ell + 1$ in $GF(2^r)$. We define the row vector $\boldsymbol{g} = (g_1, g_2, \ldots, g_\ell)$. Let $m = \left\lceil \frac{n}{\Psi(n,d)-1} \right\rceil$ and let us show that there exists a protocol P such the (U_n, T, P) is $(d,1)$-decodable and $T = T_F$ is a table with m cells. Let $H_{\boldsymbol{g}}$ be the following $m \times n$ diagonal block matrix,

$$H_{\boldsymbol{g}} = \begin{bmatrix} \boldsymbol{g} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \boldsymbol{g} \quad \boldsymbol{g'} \end{bmatrix}$$

where $\boldsymbol{g'}$ is a shortening of $\boldsymbol{g}$ to its first entries such that the matrix H has n columns. For $I = I_H, D = D_H, M = M_H$, if T_F contains a subset $S \subseteq [n]$ of size at most d, then by

[2] Note that if $k > d$ then we can append $k - d$ redundant rows to $H^d_{n,\alpha}$ and use a similar construction to obtain that $s^*(n,d,k) \leq k\log_2(n+1)$.

the definition of the insertion, deletion algorithms, each cell of T_F contains the sum of at most d different elements from a B_d-sequence. Hence, for any such S, the state of T_F is unique and there exists a listing algorithm L such that $(U_n, T_F, P = (I_H, D_H, M_H, L))$ is $(d, 1)$-decodable. ∎

Next we show that the bound in Theorem 4 is tight.

Theorem 5. Let $n = 2^r$. If the size of each cell in T is $b = \log(n)$, then $s^*(n, d, k = 1) = \left\lceil \frac{n}{\Psi(n,d)-1} \right\rceil \log(n)$.

To conclude the discussion regarding $(d, 1)$-decodable IBLT schemes, we present a construction of such a B_d-sequence in $GF(2^r)$ [2, Section V].

Construction 1. For $n = 2^r$ and an integer d, let $n'+1$ be the largest power of 2 such that $d \log(n'+1) \leq r$. Additionally, let $H^d_{n'}$ be a BCH parity-check matrix with distance at least $2d+1$ over $GF(n'+1)$. It can be verified that the sequence $g_0 = 0, g_1, \ldots, g_{n'}$, such that for $1 \leq i \leq n'$, $g_i = h_i$ is the i-th column of $H^d_{n'}$ is a B_d-sequence in $GF(n)$. Therefore, $\Psi(n, d) \geq n'+1 \geq \sqrt[d]{n}/2$.

Using Construction 1, $H_{\boldsymbol{g}}$ is a matrix with $m = \left\lceil \frac{2n}{\sqrt[d]{n}-2} \right\rceil$ rows, and thus we get the following upper bound on $s^*(n, d, k = 1)$.

Corollary 3. Let $n = 2^r$. If the size of each cell in T is $b = \log(n)$ then $s^*(n, d, k = 1) \leq \left\lceil \frac{2n}{\sqrt[d]{n}-2} \right\rceil \log(n)$.

Next we address the more intriguing case of $k = 2$. Due to space limitations, we only discuss the case where d is even. Note that for odd d, the results for $d+1$ are applicable, however, they are probably not optimal.

Construction 2. For $n = 2^r$ and an even integer d, let $n'+1$ be the largest power of 2 such that $d \log(n'+1) \leq 2r$. Additionally, let $H^d_{n'}$ be a BCH parity-check matrix with distance at least $2d+1$ over $GF(n')$ and denote by $H^{(U)}$, $H^{(L)}$ the upper, lower half of $H^d_{n'}$, respectively. As in Construction 1, let $\boldsymbol{g}^{(U)}, \boldsymbol{g}^{(L)}$ be the vector of length n' such that $g_i^{(U)}, g_i^{(L)}$, is equal to the i-th column of $H^{(U)}$, $H^{(L)}$, respectively. Define H_2 as the following block matrix

$$H_2 = \begin{bmatrix} \boldsymbol{g}^{(U)} & & & & & \\ \boldsymbol{g}^{(L)} & \boldsymbol{g}^{(U)} & & & & \\ & \boldsymbol{g}^{(L)} & \boldsymbol{g}^{(U)} & & \mathbf{0} & \\ & & \ddots & \ddots & & \\ & \mathbf{0} & & \boldsymbol{g}^{(L)} & \boldsymbol{g}^{(U)} & \\ & & & & \boldsymbol{g}^{(L)} & \boldsymbol{g}^{(U)} \\ & & & & & \boldsymbol{g}^{(L)} \end{bmatrix}.$$

Using H_2 from Construction 2, we have the following result.

Theorem 6. Let $n = 2^r$. If the size of each cell in the lookup table T is $b = \log(n)$, then for any even integer $d > 2$ we have that $s^*(n, d, k = 2) \leq \left(\left\lceil \frac{2n}{\sqrt[d]{n^2}-2} \right\rceil + 1 \right) \log(n)$.

Proof: For $H = H_2$, let $I = I_H, D = D_H$ and $M = M_H$. To prove the claim, we show that the sum of any d or less columns is unique. Let us divide the columns of H to $m-1$ disjoint sets $C_1, C_2, \ldots, C_{m-1}$ according to the location of their first non-zero entry.

First note that if all the columns are taken from the same C_i, similarly to the proof of Theorem 3, their sum is unique. Otherwise, assume to the contrary that there exist two sums of up to d columns which are the same. Since the characteristic of the field is 2, the latter implies that there exists a selection of up to $2d$ columns in H such that their sum is zero. Denote this set of columns by C. Let i be minimal such that $C \cap C_i \neq \emptyset$ and note that the i-th entry in the sum of the columns in $C \cap C_i \neq \emptyset$ must be zero. However, since the elements are taken from $\boldsymbol{g}^{(U)}$, which is a $B_{d/2}$-sequence, the latter implies that $|C \cap C_i| > d$. By the same arguments we also have that $|C \cap C_{i'}| > d$, where i' is the maximal integer for which $C \cap C_{i'} \neq \emptyset$. Thus, by recalling that $C_i \cap C_{i'} = \emptyset$, we have that $|C| > 2d$, a contradiction. ∎

In the rest of this section, we deal with the special case of $d = 4$ and show that the bound in Theorem 6 can be improved using a more sophisticated selection of the matrix H.

Construction 3. For $n = 2^r$ let $n'+1$ be the largest power of 2 such that $2\log(n'+1) \leq r$. Denote by $\widehat{H}^4_{n'}$ the matrix that is obtained from $H^4_{n'}$ (see Construction 2) by removing its left-most column. Similarly to Construction 2, let $H^{(U)}, H^{(L)}$ be the lower, upper half of $\widehat{H}^4_{n'}$ and let $\boldsymbol{g}^{(U)}, \boldsymbol{g}^{(L)}$ be the length-$(n'-1)$ vector representation of $H^{(U)}, H^{(L)}$, respectively. Define

$$\widehat{G_2} = \begin{bmatrix} \boldsymbol{g}^{(U)} & \mathbf{0} & \boldsymbol{g}^{(U)} \\ \boldsymbol{g}^{(L)} & \boldsymbol{g}^{(U)} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{g}^{(L)} & \boldsymbol{g}^{(L)} \end{bmatrix},$$

and let $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$ denote the three rows of $\widehat{G_2}$. We let $\widehat{H_2}$ be the following matrix

$$\widehat{H_2} = \begin{bmatrix} \boldsymbol{g}_1 & & & & & \\ \boldsymbol{g}_2 & & & & & \\ \boldsymbol{g}_3 & \boldsymbol{g}_1 & & & & \\ & \boldsymbol{g}_2 & & & & \\ & \boldsymbol{g}_3 & \boldsymbol{g}_1 & & & \\ & & \ddots & \ddots & & \\ & & & \boldsymbol{g}_3 & \boldsymbol{g}_1 & \\ & & & & \boldsymbol{g}_2 & \\ & & & & \boldsymbol{g}_3 & \end{bmatrix}.$$

Using Construction 3, we have the following result.

Theorem 7. Let $n = 2^r$. If the size of each cell in the lookup table T is $b = \log(n)$, then we have that

$$s^*(n, d = 4, k = 2) \leq \left(2 \left\lceil \frac{2n}{3\sqrt{n}-4} \right\rceil + 1 \right) \log(n).$$

IV. Time vs Memory Trade-Off

The results in this work demonstrate that successful listing can be achieved in the worse-case with IBLT size which is significantly smaller than the ones presented in [11]. This improvement is the result of our relaxation of the IBLT scheme to allow the use of arbitrary algorithms. It is also worth mentioning that some of the standard schemes from [11] require either additional memory to store the mapping matrix or additional time to calculate the corresponding columns during the insert and delete operations. On the contrary, for all of our constructions the only additional memory that might be needed is the primitive element α ($\log(n)$ bits) and the computation of the relevant column can be done in $O(d \log^2(n))$, which is slightly higher than the probabilistic IBLT implementations [5].

The time complexity of the listing operation of the standard and standard-indel schemes is the same as that of the probabilistic IBLT, i.e., $O(m)$ where m is the number of cells in the lookup table. However, the listing time of the constructions in Section III-C is $\max\left\{O(d\sqrt{n}), O(d^2 \log(n)\right\}$.

REFERENCES

[1] R. C. Bose and S. Chowla, "Theorems in the additive theory of numbers," North Carolina State University. Dept. of Statistics, Tech. Rep., 1960.

[2] A. E. Brouwer, J. B. Shearer, N. J. Sloane, and W. D. Smith, "A new table of constant weight codes," *IEEE Transactions on Information Theory*, vol. 36, no. 6, pp. 1334–1380, 2006.

[3] D. Eppstein and M. T. Goodrich, "Straggler identification in round-trip data streams via newton's identities and invertible Bloom filters," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 2, pp. 297–306, 2010.

[4] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, "What's the difference?: Efficient set reconciliation without prior context," in *ACM SIGCOMM*, 2011.

[5] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," *49th Annual Allerton Conference on Communication, Control, and Computing*, pp. 792–799, 2011.

[6] I. Kubjas and V. Skachek, "Partial extraction from invertible bloom filters," *IEEE International Symposium on Information Theory (ISIT)*, pp. 2415–2420, 2022.

[7] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[8] ——, "Lossradar: Fast detection of lost packets in data center networks," in *ACM International on Conference on emerging Networking EXperiments and Technologies (CoNext)*, 2016.

[9] M. Mitzenmacher and R. Pagh, "Simple multi-party set reconciliation," *Distributed Comput.*, vol. 31, no. 6, pp. 441–453, 2018.

[10] M. Mitzenmacher and G. Varghese, "Biff (Bloom filter) codes: Fast error correction for large data sets," in *IEEE International Symposium on Information Theory (ISIT)*, 2012.

[11] A. Mizrahi, D. Bar-Lev, E. Yaakobi, and O. Rottenstreich, "Invertible bloom lookup tables with listing guarantees," *arXiv preprint arXiv:2212.13812*, 2022. [Online]. Available: https://arxiv.org/abs/2212.13812

[12] K. O'Bryant, "A complete annotated bibliography of work related to Sidon sequences," *The Electronic Journal of Combinatorics [electronic only]*, 2004.

[13] A. P. Ozisik, G. Andresen, B. N. Levine, D. Tapp, G. Bissias, and S. Katkuri, "Graphene: Efficient interactive set reconciliation applied to blockchain propagation," in *ACM SIGCOMM*, 2019.

[14] R. Roth, *Introduction to Coding Theory*. Cambridge University Press, 2006.

[15] D. Schipani, M. Elia, and J. Rosenthal, "On the decoding complexity of cyclic codes up to the bch bound," *2011 IEEE International Symposium on Information Theory Proceedings*, pp. 835–839, 2011.

[16] A. T. Yaron Minsky and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Transactions on Information Theory*, vol. 49, no. 9, pp. 2213 – 2218, 2003.