Irregular Invertible Bloom Look-Up Tables

Francisco Lázaro, Balázs Matuz
Institute of Communications and Navigation of DLR (German Aerospace Center),
Wessling, Germany. Email: {Francisco.LazaroBlasco, Balazs.Matuz}@dlr.de

***Abstract*—We consider invertible Bloom lookup tables (IBLTs) which are probabilistic data structures that allow to store key-value pairs. An IBLT supports insertion and deletion of key-value pairs, as well as the recovery of all key-value pairs that have been inserted, as long as the number of key-value pairs stored in the IBLT does not exceed a certain number. The recovery operation on an IBLT can be represented as a peeling process on a bipartite graph. We present a density evolution analysis of IBLTs which allows to predict the maximum number of key-value pairs that can be inserted in the table so that recovery is still successful with high probability. This analysis holds for arbitrary irregular degree distributions and generalizes results in the literature. We complement our analysis by numerical simulations of our own IBLT design which allows to recover a larger number of key-value pairs as state-of-the-art IBLTs of same size.**

I. Introduction

Invertible Bloom lookup tables (IBLTs) were first introduced in [1] as probabilistic data structures that can be used to represent a set $\mathcal{S}$ of elements. Every element of $\mathcal{S}$ is mapped to a number d of cells of the IBLT (a formalization follows in Section II). We call an IBLT *regular* if d is a constant for every element of $\mathcal{S}$, otherwise, we say it is *irregular*. In the context of data bases the elements of $\mathcal{S}$ are key-value pairs. The key can be thought of as a short (unique) identifier of an element in the database, whereas the value is the actual data which can be orders of magnitude larger than the key. Commonly, the key associated to an element of the database is obtained simply as a hash function of its value. As the name indicates, an important property of IBLTs is that they are invertible (in contrast to Bloom filters [2]), i.e., they allow to list the elements of the set $\mathcal{S}$ which they represent. The asymptotic performance of regular IBLTs was studied in [1]. It was found that an IBLT is invertible with high probability if its *load*, defined as the ratio of key-value pairs to the number of cells, does not exceed the *load threshold*. The analysis in [1] relies on known results about the 2-core threshold of regular hypergraphs. In [3], the load threshold of specific irregular hypergraphs with only two different degrees was analyzed.

In the literature, IBLTs are applied for so-called *set reconciliation* problems aiming at establishing consistency among different sets of elements [4]. In a two party system with sets $\mathcal{S}_A$ and $\mathcal{S}_B$ one would like to determine *set differences* $\mathcal{S}_A \backslash \mathcal{S}_B$ and $\mathcal{S}_B \backslash \mathcal{S}_A$ in an efficient way and communicate the missing elements to the respective parties. Amongst others, IBLTs find applications in remote file synchronization, synchronisation of distributed databases, deduplication, or gossip protocols [5], [4]. Recently, IBLTs have been used to improve block propagation in the Bitcoin network [6].

This work, extends the analysis of irregular IBLTs. We first illustrate that the recovery operation (sometimes also referred to as inversion) of an IBLT corresponds to a peeling decoding process [7], [8], [9] on a bipartite graph. Next, we derive a density evolution analysis to obtain the load threshold. This generalizes the results of [1], [3] to arbitrary irregular IBLTs. Furthermore, we make the observation that the recovery process of IBLTs is strongly linked to the successive interference cancellation process for multiple access protocols over the collision channel [10]. Finally, we provide an irregular IBLT construction which outperforms the results in [1], [3].

This work has been accepted for presentation at the 11th International Symposium on Topics in Coding

II. Irregular invertible Bloom lookup tables

A. Description

Let $\mathcal{S} = \{z_1, z_2, ..., z_n\}$ be a set of elements, with $|\mathcal{S}| = n$. We assume that each element z is a key-value pair, denoted by $z = (x, y)$. The key x is of length ν bits and the value y is of length $\kappa \gg \nu$ bits. The key x is obtained as a function of y where the mapping is many to one. For the analysis that follows we make two simplified, but common assumptions. First, all keys x in the set are distinct, i.e., there are no key-collisions. Second, the keys x are selected uniformly from $\{0,1\}^\nu$.

Let a cell c be a data structure containing two different fields *count* and *data* where:

- *count* is an integer. It contains the number of elements that have been mapped to this cell (details on the mapping follow).
- *data* =(*data*.x, *data*.y) is a bit string of length $\nu+\kappa$ which can be divided into a pair of bit strings of length ν and κ, respectively. The bit strings *data*.x and *data*.y contain, respectively, the binary XOR of the keys and values that have been mapped to the cell.

Let us define two hash functions:

- $h_{\mathbf{\Lambda}}(x) = d$ is a non-uniform random hash function which maps an input $x \in \{0,1\}^\nu$ to an output $d \in \{1, 2, \ldots, d_{\max}\}$. The parameter $\mathbf{\Lambda} = (\Lambda_1, \Lambda_2, \ldots, \Lambda_{d_{\max}})$, referred to as *degree distribution*, is a probability mass function. Under the assumption that the input x is uniformly distributed, we have $P(d = i) = \Lambda_i$, i.e., the output of $h_{\mathbf{\Lambda}}(x)$ follows the degree distribution $\mathbf{\Lambda}$.
- $H_{m,d}(x) = \boldsymbol{g}$ is a random hash function which maps an input $x \in \{0,1\}^\nu$ to a length-d vector $\boldsymbol{g}$ of d *different* natural numbers in $\{1, 2, \ldots, m\}$, i.e., it samples

Algorithm 1 Initialization

procedure INITIALIZE()
  for i = $1, 2, \ldots, m$ **do**
    $c_i.count = 0$
    $c_i.data = \mathbf{0}$

Algorithm 2 Insertion

procedure INSERT(z)
  $d \leftarrow h_{\mathbf{\Lambda}}(z.x)$
  $\boldsymbol{g} \leftarrow H_{m,d}(z.x)$
  for i = $1, 2, \ldots, d$ **do**
    $c_{g_i}.count = c_{g_i}.count + 1$
    $c_{g_i}.data = \text{XOR}\,(c_{g_i}.data,\ z)$

Algorithm 3 Deletion

procedure DELETE(z)
  $d \leftarrow h_{\mathbf{\Lambda}}(z.x)$
  $\boldsymbol{g} \leftarrow H_{m,d}(z.x)$
  for i = $1, 2, \ldots, d$ **do**
    $c_{g_i}.count = c_{g_i}.count - 1$
    $c_{g_i}.data = \text{XOR}\,(c_{g_i}.data, z)$

Algorithm 4 Recovery

procedure RECOVER()
  while $\exists i \in [1, m] | c_i.count = 1$ **do**
    add $z = c_i.data$ to the output list
    call Delete (z)

d different natural numbers between 1 and m *without replacement*. Such a hash function can be obtained from a uniform random hash function that outputs a natural number between 1 and $\prod_{i=0}^{d-1}(m-i)$.

An irregular IBLT is a probabilistic data structure to store elements of a set $\mathcal{S}$. It is defined by its degree distribution $\mathbf{\Lambda}$, the number of cells (or length) m, and the random hash functions $h_{\mathbf{\Lambda}}(x)$ and $H_{m,d}(x)$. An IBLT supports several operations: initialization, insertion, deletion, and recovery:

- Initialize(). This operation sets the different fields of all the cells in the IBLT to zero.
- Insert(z). The insertion operation *adds* the key-value pair z to the IBLT (see Algorithm 2).
- Delete(z). The deletion operation *removes* the key-value pair z to the IBLT (see Algorithm 3).
- Recover(). This operation aims at outputting all the key-value pairs stored in the IBLT. If this operation provides the full list of key-value pairs in the IBLT, we say it succeeded. Otherwise, if it provides an incomplete list, we say the list operation fails (see also Algorithm 4).

B. Encoding $\mathcal{S}$ into an IBLT

The mapping of the n elements of $\mathcal{S}$ to an IBLT, also referred to as encoding is done as follows. First, all cells are initialized to zero as described by Algorithm 1. After initialization, the elements of $\mathcal{S}$ are successively inserted into the IBLT as described by Algorithm 2: for every element $z = (x, y)$, $d = h_{\mathbf{\Lambda}}(x)$ cells with indices $H_{m,d}(x) = \boldsymbol{g}$ are selected. The element z is then XOR-ed with the *data* field of the cells, and their *count* field is increased by one.

C. Recovery of $\mathcal{S}$

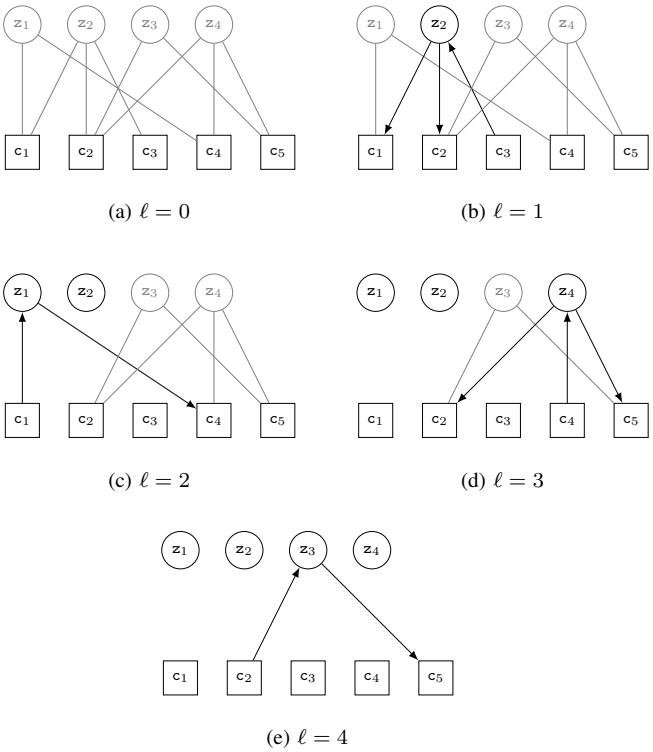
We are interested in recovering all n elements of $\mathcal{S}$ from the irregular IBLT of length m. This process is also referred to as recovery and or decoding. Recovery succeeds if all m cells of the IBLT have *count* field equal to zero. In this case the output of the recovery operation will contain all n elements that had been inserted. Otherwise, if some cells have a non-zero count, recovery fails. A low-complexity algorithm for the recovery of IBLTs was proposed in [1], instantiated for a regular IBLT. Algorithm 4 describes the recovery operation for an irregular IBLT. We seek for cells with counter field equal to one, since the *data* field of such cells is an element z of $\mathcal{S}$. Then, $z = (x, y)$ is deleted from the IBLT by calling Delete(z), which removes z from $h_{\mathbf{\Lambda}}(x) = d$ cells with indices $H_{m,d}(x) = \boldsymbol{g}$. Since successful recovery requires processing all n elements of $\mathcal{S}$, and each element gets mapped in average to $\bar{d}$ different cells, complexity of the recovery operations scales as $\mathcal{O}(n\bar{d})$ (which is the same as the encoding complexity).

D. Peeling decoding

We argue that the recovery operation is an instance of *peeling decoding* [8]. We may represent an IBLT as a bipartite (or Tanner) graph $\mathcal{G} = (\mathcal{Z} \cup \mathcal{C}, \mathcal{E})$ composed of a set of n data nodes $\mathcal{Z}$, a set of m cell nodes $\mathcal{C}$ and a set of edges $\mathcal{E}$. As the names indicate, data nodes represent key-value pairs and cell nodes represent cells of the IBLT. A data node $\mathtt{z}_i \in \mathcal{Z}$ and a cell node $\mathtt{c}_h \in \mathcal{C}$ are connected by an edge if and only if $z_i = (x_i, y_i)$ is written to cell c_h, i.e., $\exists k | g_k = h$, where $\boldsymbol{g} = H_{m,d}(x_i)$ and $d = h_{\mathbf{\Lambda}}(x_i)$. A data node $\mathtt{z}$ and a cell node $\mathtt{c}$ are said to be neighbors if they are connected by an edge. We use the shorthand $\mathtt{z} \in \mathcal{N}(\mathtt{c})$ or $\mathtt{c} \in \mathcal{N}(\mathtt{z})$. The degree of a node is given by the number of edges connected to the node. Thus, the degree of a cell node equals the *count* field of the cell it represents.

Recovery of $\mathcal{S}$ can be represented as a peeling process on a bipartite graph where the graph is *unknown to the decoder* and is revealed during the decoding process. In particular, whenever a cell node $\mathtt{c}$ of degree one is present, its only neighbor $\mathcal{N}(\mathtt{c}) = \mathtt{z}$ is determined. The key-value pair z which is represented by the data node $\mathtt{z}$ is added to the output list. Next, the retrieved key-value pair is removed from the IBLT which translates into the removal of all edges attached to its associated data node. This process is repeated until no more cell nodes of degree one are present. At this stage, if all cell nodes are of degree zero, recovery succeeded and all key-value pairs are present in the output list. Otherwise, if some cell nodes of degree larger than zero are present, recovery fails, and the output list will not contain all key-value pairs.

Example 1 (Peeling decoding)**.** *The different steps of the peeling process are shown in Figure 1. Figure 1a shows the bipartite graph representation of an IBLT before the peeling*



(a) $\ell = 0$ (b) $\ell = 1$

(c) $\ell = 2$ (d) $\ell = 3$

(e) $\ell = 4$

Fig. 1: Peeling process on the graph representation of an IBLT with $n = 4$ key-value pairs and $m = 5$ cells. The index ℓ is used to denote the different steps of the recovery process.

process starts. We observe that the IBLT has $m = 5$ cell and stores $n = 4$ key-value pairs. However, at this stage the depicted bipartite graph is unknown to the decoder, since it does not have any knowledge about $\mathcal{S}$. The decoder is only aware of the m cell nodes. For this reason, the data nodes as well as the edges are shown in grey. The graph structure will be revealed successively as the recovery operation progresses, and it will only be completely known if decoding succeeds. Otherwise, a part of the graph will remain hidden. We can see that cell node $\mathtt{c}_3$ has degree 1, and thus its associated IBLT cell c_3 has count 1. The recovery operation retrieves the only key-value pair that has been mapped to cell c_3, i.e., data node z_2, which is added to the output list of the recovery operation. Afterwards, z_2 is deleted from the IBLT. In the graph representation this translates to revealing the only neighbor of cell node $\mathtt{c}_3$, data node $\mathtt{z}_2$ (now shown in black), and deleting all edges attached to it, as shown in Figure 1b. As a consequence, the degree of $\mathtt{c}_1$ becomes one. In the next step, as shown in Figure 1c, the only neighbor of $\mathtt{c}_1$, $\mathtt{z}_1$, is revealed and all edges attached to it are removed. This reduces the degree from $\mathtt{c}_4$ from 2 to 1. Then, data node $\mathtt{z}_4$ is revealed since it is the only neighbor of $\mathtt{c}_4$. After all edges attached to $\mathtt{z}_4$ are removed, as shown in Figure 1d, we have two cell nodes of degree 1, namely $\mathtt{c}_2$ and $\mathtt{c}_5$, both of which have as only neighbor $\mathtt{z}_3$. In the last step shown in Figure 1e, first $\mathtt{z}_3$ is revealed as the only neighbor of $\mathtt{c}_2$. Finally, all edges attached to $\mathtt{z}_3$ are erased from the graph. In this example recovery operation succeeded and set $\mathcal{S}$ was completely recovered.

III. Analysis of the recovery process

A. Degree Distributions

Let us define the node perspective degree distribution polynomial for the data nodes as

$$\Lambda(\mathrm{x}) = \sum_{d=1}^{d_{\max}} \Lambda_d \, \mathrm{x}^d$$

where x is a dummy variable and Λ_d is the probability of a data node $\mathtt{z}$ being of degree d. Similarly, the node perspective degree distribution polynomial for the cell nodes is

$$\Psi(\mathrm{x}) = \sum_{d=0}^{n} \Psi_d \, \mathrm{x}^d$$

where Ψ_d corresponds to the probability of a cell node $\mathtt{c}$ having degree d. In literature, $\Lambda(x)$ and $\Psi(x)$ are sometimes referred to as *left* and *right* degree distributions, a convention that has its origins in LDPC literature. It is easy to verify that the average node degrees are obtained by evaluating the derivative of the polynomials in $\mathrm{x} = 1$, i.e., $\Lambda'(1)$, and $\Psi'(1)$ respectively.

Note that the data node degree distribution $\Lambda(\mathrm{x})$ is a design parameter while the cell node distribution $\Psi(\mathrm{x})$ is induced by $\Lambda(\mathrm{x})$, the number of cells m and the cardinality n of $\mathcal{S}$. In particular, observe that the number of edges connected to the n data nodes must be the same as the number of those connected the m cell nodes, i.e.,

$$n\Lambda'(1) = m\Psi'(1).$$

Since $d = h_{\boldsymbol{\Lambda}}(x)$ follows the probability distribution $P(d = i) = \Lambda_i$, and $\boldsymbol{g} = H_{m,d}(x)$ is a length-d vector of different numbers between 1 and m chosen uniformly at random without replacement, the probability that a data node $\mathtt{z}$ is connected to a given cell node $\mathtt{c}$ is

$$\mathrm{P}\{\mathtt{c} \in \mathcal{N}(\mathtt{z})\} = \frac{\Psi'(1)}{n}.$$

If we assume that the outputs of the hash functions $h_{\boldsymbol{\Lambda}}(x)$ and $H_{m,d}(x)$ are independent for different inputs, then the probability that a cell node $\mathtt{c}$ is connected to d data nodes follows a binomial distribution,

$$\Psi_d = \binom{n}{d} \left(\frac{\Psi'(1)}{n}\right)^d \left(1 - \frac{\Psi'(1)}{n}\right)^{n-d}.$$

Instead of the node-perspective degree distributions, one may also use edge-perspective degree distributions. Let us define by λ_d (and ρ_d) the probability that a generic edge in the bipartite graph is connected to a degree d data node (a degree d cell node). We have

$$\lambda_d = \frac{\Lambda_d d}{\sum_\ell \Lambda_\ell \ell} \qquad \text{and} \qquad \rho_d = \frac{\Psi_d d}{\sum_\ell \Psi_\ell \ell}.$$

For convenience, the polynomial representations of λ_d and ρ_d are chosen to be

$$\lambda(\mathrm{x}) = \sum_d \lambda_d \mathrm{x}^{d-1} \qquad \text{and} \qquad \rho(\mathrm{x}) = \sum_d \rho_d \mathrm{x}^{d-1}.$$

B. Density Evolution

Let us define the *load* $\eta = n/m$ as the ratio between the number of key-value pairs and cells, and let us consider the regime in which n and m tend to infinity while keeping the load η constant. For a given $\lambda(\mathrm{x})$ and load η we are interested in determining whether the recovery operation will be successful or not. In literature, the performance of peeling decoding is analyzed via *density evolution* [8], [11], which restates the peeling decoder as an *equivalent* iterative message passing algorithm where nodes pass messages along the edges to their neighbors. In our case, the messages exchanged by the nodes can be either an *erasure*, i.e., we do not know the corresponding key-value pair yet, or the opposite, *non-erasure*, meaning that key-value pair has been recovered. In particular, given an ensemble of bipartite graphs $\mathscr{G}(n, \eta, \lambda)$ with n data nodes, $m = n/\eta$ cell nodes, and edge oriented degree distribution $\lambda(\mathrm{x})$, density evolution yields the average probability of the exchanged messages at the ℓth iteration being an erasure assuming that n goes to infinity.

Denote by p_ℓ the (average) probability that the message sent from a cell node over an edge at the ℓth iteration is an erasure and by q_ℓ the (average) probability that the message sent from a data node over an edge at the ℓth iteration is an erasure. Consider first the message sent by a cell node of degree d over a given edge. This message will be a non-erasure if the messages received through the remaining $d-1$ edges were non-erasure messages. Thus we have

$$\begin{aligned} 1 - p_\ell &= (1 - q_\ell)^{d-1} \\ p_\ell &= 1 - (1 - q_\ell)^{d-1}. \end{aligned}$$

Similarly, if we consider a data node of degree d, the message sent over an edge will be an erasure only if all messages received over all other $d-1$ edges were erasures. Thus,

$$q_\ell = {p_{\ell-1}}^{d-1}.$$

We are interested in the average erasure probability, where the average is taken over all edges of all bipartite graphs in $\mathscr{G}(n, \eta, \lambda)$, hence we have

$$q_\ell = \sum_d \lambda_d {p_{\ell-1}}^{d-1} = \lambda(p_{\ell-1}). \tag{1}$$

Similarly, the average probability that a message sent by a cell node over a random edge is an erasure can be obtained as

$$p_\ell = \sum_d \rho_d \left(1 - (1 - q_\ell)^{d-1}\right) = 1 - \rho(1 - q_\ell). \tag{2}$$

Initially, we have $q_0 = p_0 = 1$, i.e., we start by setting all messages to erasures. Then, by iteratively applying (1) and (2) we can track the evolution of q_ℓ and p_ℓ as the number of iterations ℓ grows. Note that q_ℓ corresponds to the probability that a randomly chosen key-value pair has been recovered after ℓ iterations.

As shown in [8], the probability of non-erasure (i.e., success) is subject to a threshold effect (or phase transition) at $\eta = \eta^*$, referred to as load threshold in the sequel. In particular, the list operation will be successful with probability tending to 1 for loads η fulfilling $0 < \eta \leq \eta^*$. According to [8], the load threshold η^* can be formally expressed as the maximum value of η for which

$$q > \lambda(1 - \rho(1 - q)), \qquad \forall q \in (0, 1]. \tag{3}$$

Note that the dependency on η is implicit in $\rho(\mathrm{x})$. In particular, in the asymptotic regime when $n \to \infty$, we can express $\Psi(\mathrm{x})$ as

$$\Psi(\mathrm{x}) = e^{-\Psi'(1)(1-\mathrm{x})} = e^{-\eta\,\Lambda'(1)\,(1-\mathrm{x})}$$

which allows to rewrite $\rho(\mathrm{x})$ as

$$\rho(\mathrm{x}) = \frac{\Psi'(\mathrm{x})}{\Psi'(1)} = e^{-\eta\,\Lambda'(1)\,(1-\mathrm{x})}. \tag{4}$$

Substituting $\rho(\mathrm{x})$ in (3) by (4) yields

$$q > \lambda\left(1 - e^{\eta\Lambda'(1)q}\right), \qquad \forall q \in (0, 1]$$

which explicitly shows the dependency on η.

C. Connection to IRSA

For the bipartite graphs used to represent IBLTs the left degree distribution $\Lambda(\mathrm{x})$ (or $\lambda(\mathrm{x})$) is a free parameter whereas the right degree distribution $\Psi(\mathrm{x})$ corresponds to a binomial distribution. Such bipartite graphs have been studied in depth in the context of a random access protocol known as irregular repetition coded slotted ALOHA (IRSA) over the collision channel [10]. A few important results on such graphs are listed in the following. The asymptotic regime was first studied in [10], where a density evolution analysis was presented. In [12] a sequence of capacity achieving degree distributions was presented, i.e., ensembles whose load threshold converges to $\eta^* = 1$. For the finite length regime, an approximate error-floor analysis was presented in [13] whereas an approximate analysis of the waterfall performance was presented in [14].

IV. Numerical Results

Table I shows the load thresholds η^* for different regular and irregular data node degree distributions obtained via density evolution. For regular distributions, we observe that the load thresholds obtained with the analysis in Section III coincide with the thresholds reported in [1], where a different technique was used to obtain the thresholds.[1] Among the regular distributions, $\Lambda(\mathrm{x}) = \mathrm{x}^3$ yields the best threshold $\eta^* = 0.8183$.

In addition to regular distributions, Table I also provides the load thresholds for three irregular distributions whose load thresholds are higher than those of regular distributions. The slightly irregular distribution $0.887\mathrm{x}^3 + 0.113\mathrm{x}^{21}$ with threshold 0.92 is taken from [3], where it was conjectured to be the best irregular distribution with two degrees. The distribution $0.25\mathrm{x}^2 + 0.6\mathrm{x}^3 + 0.15\mathrm{x}^8$ for IRSA is taken from [10], and was designed to exhibit good performance for moderate values of m. Additionally, following the analysis in Section III we derive the degree distribution $0.15\mathrm{x}^2 + 0.725\mathrm{x}^3 + 0.125\mathrm{x}^{18}$ with threshold 0.934 by using an optimization algorithm called simulated annealing. In particular, the goal of the optimization was maximizing the load threshold, see (3), while limiting the probability of degree 2 since it is associated with high error floors for small and moderate values of m [13].

TABLE I: Load thresholds η^* for different degree distributions

$\Lambda(x)$	η^*
x^3	0.818
x^4	0.772
$0.887x^3 + 0.113x^{21}$	0.920
$0.25x^2 + 0.6x^3 + 0.15x^8$	0.892
$0.15x^2 + 0.725x^3 + 0.125x^{18}$	0.934

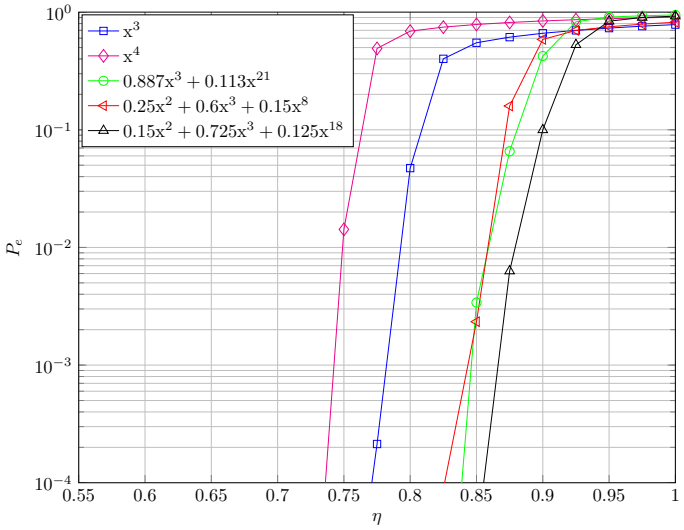


Fig. 2: Probability of unsuccessful of a key-value pair, P_e, as a function of η for different regular and irregular data node degree distributions with $m = 2000$.

Monte Carlo simulations to determine the probability of a key-value pair not being recovered (not present in the output list of the recovery operation), termed P_e, versus the channel load η are shown in Figure 2. We simulated IBLTs with $m = 2000$ for the different degree distribution in Table I. If we compare the curves in the figure with the asymptotic load thresholds in Table I, we observe that the load threshold provides a good estimate of the load for which P_e undergoes a phase transition, i.e., for which P_e shows a sharp drop. The irregular distributions outperform their regular counterparts. Among the presented distributions, $\Lambda(x) = 0.15x^2 + 0.725x^3 + 0.125x^{18}$ found by simulated annealing yields the best performance.

V. Conclusion and Outlook

In this paper we discuss degree distributions for irregular IBLTs. Realizing recovery corresponds to peeling decoding, we provide a density evolution analysis, which is a novel tool to analyze IBLTs and extends results from the literature. Furthermore, we show that the graphs induced by IBLTs, are characterized by a binomial right degree distribution, a family of graphs which has been studied in the framework of random access protocols. This allows to borrow powerful tools from the literature for future work on IBLTs. Finally, using density evolution we design a degree distribution which outperforms known distributions for IBLTs.

Despite the fact that the bipartite graphs arising from IBLTs have been studied in practice in the context of IRSA, some questions related to IBLTs still remain open. First, in the context of IRSA the interesting regime is that of small or moderate values of m, due to latency constraints. Also, owing to energy efficiency considerations, IRSA distributions usually feature a low average and maximum degree. For IBLTs scenarios with larger m, larger average and maximum degrees might be of interest. Second, and more importantly, for some applications [4], at the time in which the size of the IBLT is fixed, the number of key-value pairs which will be inserted in it is not known or deviates strongly from its estimated value. So far, schemes based on IBLTs solve this by oversizing the IBLT, i.e. operating at lower loads, which is inefficient. A more advantageous scheme would be one that allows to add IBLT cells on demand, similarly as it is done in frameless ALOHA [15].

[1] In [1] results are reported in terms of $1/\eta$.

Acknowledgements

The authors would like to thank Federico Clazzer for providing the software used for the Monte Carlo simulations.

References

[1] M. Goodrich and M. Mitzenmacher, "Invertible Bloom lookup tables," in *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. Monticello, IL, USA: IEEE, 2011, pp. 792–799.

[2] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[3] M. Rink, "Mixed hypergraphs for linear-time construction of denser hashing-based data structures," in *Proc. of the Int. Conf. on Current Trends in Theory and Practice of Comp. Science*. Springer, 2013, pp. 356–368.

[4] D. Eppstein, M. Goodrich, F. Uyeda, and G. Varghese, "What's the difference?: Efficient set reconciliation without prior context," *ACM SIGCOMM Comp. Commun. Review*, vol. 41, no. 4, pp. 218–229, 2011.

[5] M. Mitzenmacher and R. Pagh, "Simple multi-party set reconciliation," *Distributed Comput.*, vol. 31, no. 6, pp. 441–453, 2018. [Online]. Available: https://doi.org/10.1007/s00446-017-0316-0

[6] P. Ozisik, G. Andresen, B. Levine, D. Tapp, G. Bissias, and S. Katkuri, "Graphene: Efficient interactive set reconciliation applied to Blockchain propagation," in *Proc. of Conf. of the ACM Special Interest Group on Data Commun.* Beijing, China: ACM, Aug. 2019, pp. 303–317.

[7] N. Alon, J. Edmonds, and M. Luby, "Linear time erasure codes with nearly optimal recovery," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*. Milwaukee, WI, USA: IEEE, Oct. 1995, pp. 512–519.

[8] M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proc. of the 9-th annual ACM-SIAM Symp. on Discrete Algs.* San Francisco, CAL, USA: ACM, 1998, pp. 364–373.

[9] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Trans. on Inf. Theory*, vol. 47, no. 2, pp. 569–584, Feb. 2001.

[10] G. Liva, "Graph-based analysis and optimization of contention resolution diversity slotted ALOHA," *IEEE Trans. on Commun.*, vol. 59, no. 2, pp. 477–487, 2011.

[11] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. on Inf. Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[12] K. Narayanan and H. Pfister, "Iterative collision resolution for slotted ALOHA: An optimal uncoordinated transmission policy," in *Proc. of 7th Int. Symp. on Turbo Codes and Iterative Inf. Processing (ISTC)*. Gothenburg, Sweden: IEEE, 2012, pp. 136–139.

[13] M. Ivanov, F. Brännström, A. Graell i Amat, and P. Popovski, "Error floor analysis of coded slotted ALOHA over packet erasure channels," *IEEE Commun. Letters*, vol. 19, no. 3, pp. 419–422, 2015.

[14] A. Graell i Amat and G. Liva, "Finite-length analysis of irregular repetition slotted ALOHA in the waterfall region," *IEEE Commun. Letters*, vol. 22, no. 5, pp. 886–889, 2018.

[15] F. Lázaro, C. Stefanović, and P. Popovski, "Reliability-latency performance of frameless ALOHA with and without feedback," *IEEE Trans. Commun.*, vol. 68, no. 10, pp. 6302–6316, Jul. 2020.