How to Store a Random Walk

Emanuele Viola* Omri Weinstein† Huacheng Yu‡

Abstract

Motivated by storage applications, we study the following data structure problem: An encoder wishes to store a collection of jointly-distributed files $\overline{X} := (X_1, X_2, \ldots, X_n) \sim \mu$ which are *correlated* ($H_\mu(\overline{X}) \ll \sum_i H_\mu(X_i)$), using as little (expected) memory as possible, such that each individual file X_i can be recovered quickly with few (ideally constant) memory accesses.

In the case of independent random files, a dramatic result by Pătraşcu (FOCS'08) and subsequently by Dodis, Pătraşcu and Thorup (STOC'10) shows that it is possible to store $\overline{X}$ using just a *constant* number of extra bits beyond the information-theoretic minimum space, while at the same time decoding each X_i in constant time. However, in the (realistic) case where the files are correlated, much weaker results are known, requiring at least $\Omega(n/\mathrm{poly}\lg n)$ extra bits for constant decoding time, even for "simple" joint distributions μ.

We focus on the natural case of compressing *Markov chains*, i.e., storing a length-n random walk on any (possibly directed) graph G. Denoting by $\kappa(G, n)$ the number of length-n walks on G, we show that there is a succinct data structure storing a random walk using $\lg_2 \kappa(G, n) + O(\lg n)$ bits of space, such that any vertex along the walk can be decoded in $O(1)$ time on a word-RAM. If the graph is strongly connected (e.g., undirected), the space can be improved to only $\lg_2 \kappa(G, n) + 5$ extra bits. For the harder task of matching the *point-wise* optimal space of the walk, i.e., the empirical entropy $\sum_{i=1}^{n-1} \lg(deg(v_i))$, we present a data structure with $O(1)$ extra bits at the price of $O(\lg n)$ decoding time, and show that any improvement on this would lead to an improved solution on the long-standing Dictionary problem. All of our data structures support the *online* version of the problem with constant update and query time.

*Northeastern University. Email: viola@ccs.neu.edu. Supported by NSF CCF award 1813930.

†Columbia University. Email: omri@cs.columbia.edu. Research supported by NSF CAREER award CCF-1844887.

‡Harvard University. Email: yuhch123@gmail.com. Supported in part by ONR grant N00014-17-1-2127, a Simons Investigator Award and NSF Award CCF 1715187. This work was initiated during a visit of the authors to the Simons Institute.

1 Introduction

Consider the following information-retrieval problem: an encoder (say, a Dropbox server) receives a large collection of jointly-distributed files $\overline{X} := (X_1, X_2, \ldots, X_n) \sim \mu$ which are highly *correlated*, i.e.,[1]

$$H_\mu(\overline{X}) \ll \sum_i H_\mu(X_i), \tag{1}$$

and needs to preprocess $\overline{X}$ into as little (expected) memory as possible, such that each individual file X_i can be retrieved quickly using few (ideally constant) memory accesses. This data structure problem has two naïve solutions: The first one is to compress the entire collection using Huffman (or Arithmetic) coding, achieving essentially optimal space $s = H_\mu(\overline{X}) + 1$ bits (in expectation), but such entropy-coding schemes generally require decompressing the *entire* file even if only a single file X_i needs to be retrieved. The other extreme is to compress each file separately, achieving (highly) suboptimal space $s \sim \sum_i H_\mu(X_i) + n \gg H_\mu(\overline{X})$ at the upshot of constant decoding time $t = O(1)$. Is it possible to get the best of both worlds (time and space), i.e., a *locally-decodable* data compression scheme? [2]

This problem is motivated by large-scale storage applications. The proliferation in digital data being uploaded and analyzed on remote servers is posing a real challenge of scalability in modern storage systems. This challenge is incurred, in part, by the redundancy of maintaining very large yet low-entropy datasets. At the same time, in many modern storage applications such as genome sequencing and analysis, real-time financial trading, image processing etc., databases are no longer merely serving archival purposes – data needs to be continually accessed and manipulated for training, prediction and real-time statistical decision making [TBW18, HBB+18, GPL+00, THOW16, CS00]. This inherent tension between *compression and search*, i.e., the need to perform local computations and search over the compressed data itself without first decompressing the dataset, has motivated the design of compressions schemes which provide *random access* to individual datapoints, at a small compromise of the compression rate, giving rise to the notion of *locally-decodable source coding* (LDSC) [Pat08, DLRR13, MHMP15, MCW15, TBW18]. Local decodability is also a crucial aspect in distributed file systems, where the energy cost of random-accessing a memory address is typically much higher than that of sending its actual content, especially in SSD hardware [APW+08].

There is a long line of work successfully addressing the LDSC problem in *i.i.d* or nearly-i.i.d settings (i.e., when (1) holds with equality or up to $\pm o(n)$ additive factor [Pag02, BMRV02, GRR08, Pat08, DPT10, MCW15, MHMP15, TBW18, BN13]), where *succinct* data structures are possible. In contrast, the *correlated* case (1) is much less understood (more on this in Sections 1.3 and 4.2). Clearly, the aforementioned tradeoff between compression and search is all the more challenging (yet appealing) when the dataset exhibits many similarities/correlations (i.e., $H_\mu(\overline{X}) \ll \sum_i H_\mu(X_i)$), and this is indeed the reality of a large portion of digital media [WSY+16]. Once again, joint-compression of the entire dataset is information-theoretically superior in terms of minimizing storage space, but at the same time, global compression, by nature, extinguishes any "locality" in the original data, making it useless for random-access applications.

Unfortunately, a simple observation (see Proposition 1) shows that the general LDSC problem in (1) is a "complete" static data structure problem (in the cell-probe model), in the sense that any data structure problem $\mathcal{P}$ (with an *arbitrary* set of n database queries) can be "embedded" as an LDSC problem on *some* joint distribution $\mu = \mu(\mathcal{P})$. This observation implies that locally-decodable data compression is generally impossible, namely, for *most* low-entropy distributions μ on n files ($H_\mu := H_\mu(\overline{X}) \ll n$), any data structure requires either near-trivial storage space $s \gtrsim n^{1-o(1)}$ or decoding time $t \geq (H_\mu)^{1-o(1)}$ for each file (An *explicit* hard example is the family of (H_μ)-wise independent distributions, for which

[1] $H_\mu(X)$ denotes the Shannon entropy of $X \sim \mu$

[2] Of course, one could consider a combination of the two solutions by dividing the files into blocks of size $\sim t$, and compressing each one optimally, but this method can in general be arbitrarily lossy in entropy (space) when the blocks are correlated.

our reduction implies $t \geq \Omega(\lg H_\mu)$ retrieval time unless super-linear $s = \omega(H_\mu)$ storage space is used). These unconditional impossibility results naturally raise the question:

> Which statistical properties of low-entropy distributions μ facilitate an efficient LDSC scheme with space $O(H_\mu(X))$, in the word-RAM model?

Perhaps the most basic family of low-entropy joint distributions is that of *Markov chains*, which models "time-decaying" correlations in discrete-time sequences. Ergodicity is a standard statistical model in many realistic datasets (e.g., financial time-series, DNA sequences and weather forecasting to mention a few [THOW16, CS00, GY07]). We study the LDSC question on Markov chains, captured by the problem of storing a length-n *random walk* on (directed or undirected) graphs of finite size. Our data strucutres greatly improve on previously known time-space tradeoffs for Markov chains obtained via "universal compression" (e.g. [DLRR13, TBW18], see elaboration in Section 1.3), and also efficiently support the dynamic version of the problem where the walk evolves step-by-step in an online fashion. Throughout the paper, all logarithms are base 2 unless otherwise stated.

1.1 Main Results

To build intuition for the problem, let G be an undirected d-regular graph of finite (but arbitrarily large constant) size, and consider a random walk $W = (V_0, V_1, V_2, \ldots, V_n)$ on G starting from a random vertex (for simplicity), where $V_i \in [|G|]$ denotes the ith vertex v_i visited in the walk. Excluding the first vertex V_0 for convenience of notation, the joint entropy of W is clearly $H(W) = \lg|G| + n\lg d$, whereas the sum-of-marginal entropies is

$$\sum_{i=0}^{n} H(V_i) = (n+1)\lg|G| \gg n\lg d \tag{2}$$

since each V_i is marginally uniform (as G was assumed to be regular and we started from a uniform vertex). This simple instance already captures an interesting case of the LDSC problem: The information-theoretic minimum space for storing the walk ($\lg|G| + n\lg d$ bits) can be achieved by storing for each vertex v_i in the walk the next outgoing edge (when d is not a power of 2 we can use arithmetic coding to compress the entire sequence jointly with 1 bit loss in space). In either case, such encoding does not enable random access: retrieving v_i requires "unfolding" the walk all the way to the start, taking $t \sim \Omega(n)$ time in the worst case (by preprocessing how to unfold consecutive $\lg n$ steps, the decoding time can be improved to $\Omega(n/\lg n)$). The other extreme is to store the walk explicitly (storing V_i using $\lceil\lg|G|\rceil$ bits), facilitating $O(1)$ decoding time, at the price of highly redundant space (at least the LHS of (2)). Of course, it is possible to combine the two approaches and compress each t-step subwalk optimally, losing at most $\sim \lg(|G|/d)$ bits per block, so that decoding time is $O(t/w)$ and storage is suboptimal by an additive term of $r = O_{|G|}(n/t)$ bits, where r is the *redundancy* of the data structure. This linear tradeoff implies, for example, that if we wish to decode each vertex in constant time, the redundancy of the data structure is $r \sim \frac{n\lg(|G|/d)}{\lg n}$ on a word-RAM with word-size $w = \Theta(\lg n)$.

We show that at the price of $r = 3$ extra bits of redundancy, each vertex in the walk can be decoded in constant time, when the underlying graph is d-regular for any d (not necessarily a power of 2):

Theorem 1 (Informal)**.** *Given a walk $(v_0, \ldots, v_n)$ in G, there is a succinct* cell-probe *data structure using $\lg|G| + n\lg d + 3$ bits of memory, supporting retrieval of any vertex v_i ($i \in [n]$) in constant time, assuming word-size $w \geq \Omega(\lg n)$. Moreover, the data structure can be implemented on a word RAM, assuming a precomputed[3] ℓ-bit look-up table, supporting vertex retrievals in $O(\frac{\lg\lg n}{\lg\lg \ell})$ time. The constants in the query times depend on the size of G.*

[3]This lookup table only contains precomputed information about the graph G and does not depend on the input walk $(v_0, \ldots, v_n)$. If we are storing multiple walks, it can be shared across instances, hence has a small "amortized" cost.

Dealing with general (directed, non-regular) graphs is fundamentally more challenging, the main reason being that: (a) the stationary distribution of non-regular graphs is not uniform; (b) the number of sub-walks between any two vertices (u, v) is not fixed when the graph is non-regular or directed, i.e., the distribution on sub-walks is again nonuniform. (We elaborate why uniformity is crucial for succinct solutions and how we resolve this challenge in the following Section 1.2). Intuitively, this means that if we want our space to (roughly) match the entropy of the walk, then for non-regular graphs *variable-length* coding must be used, making decoding in constant-time very challenging (as the memory locations corresponding to substrings become unpredictable).

To this end, it is natural to consider two space benchmarks for general graphs. The first benchmark is the *point-wise* optimal space for storing the walk $v = (v_0, \ldots, v_n)$, i.e., its empirical entropy

$$\lg |G| + \sum_{i=0}^{n-1} \lg(\deg_G(v_i)), \tag{3}$$

which is clearly the information-theoretic minimum space (as the expectation of this term over a random walk V is precisely the Shannon entropy of the walk). A somewhat more modest *worst-case* space benchmark (depending on how "non-regular" the graph G is), is

$$\lg \kappa(G, n) = \lg(\mathbf{1}^\top {A_G}^n \mathbf{1}) \tag{4}$$

where $\kappa(G, n)$ is the number of length-n walks on G and A_G is the adjacency matrix of G. Note that for regular graphs the two benchmarks (3) and (4) are the same, hence Theorem 1 is best possible.

Our first result for general graphs, building on the work of [Pat08], is a data structure with $O(1)$ extra bits beyond the point-wise optimal space, and $O(\lg n)$ decoding time on a word-RAM.

Theorem 2 (Point-wise compression for general graphs, Informal)**.** *Given a length-n walk $(v_0, \ldots, v_n)$ on any directed graph G, there is a succinct data structure that uses $\lg_2 |G| + \sum_{i=0}^{n-1} \lg(\deg(v_i)) + O(1)$ bits of space, and retrieves any vertex v_i in $O(\lg n)$ time on a word-RAM, assuming a precomputed[4] lookup-table of size $\ell = \tilde{O}(n^6)$. The constant in the query time depends on the size of G.*

Nevertheless, we show that if one is willing to match the *worst-case* space benchmark (4), then *constant* query time is possible for any graph G. This is our main result and the technical centerpiece of the paper. We state this result in its most general form.

Theorem 3 (Worst-case compression for general graphs, Informal)**.** *Given a length-n walk on any strongly connected directed graph G, there is a data structure that uses $\lg(\mathbf{1}^\top A^n \mathbf{1}) + O(1)$ bits of space, and retrieves any vertex v_i in $O(\frac{\lg \lg n}{\lg \lg \ell})$ time on a word-RAM, assuming a precomputed lookup table of size ℓ. For general directed graphs, the same holds albeit with $O(\lg n)$ bits of redundancy. The constant in the query time depends on the size of G.*

It is natural to ask whether constant-time decoding is possible even with respect to the point-wise space benchmark. Our final result shows that any improvement on Theorem 2 would lead to an improvement on the long-standing *succinct Dictionary* problem [Pag02, Pat08]: We present a succinct reduction from Dictionary to storing a random walk on some non-regular (directed) graph, under the technical restriction that all marginal symbol frequencies in the input dictionary string are powers of two [5].

Theorem 4. *Let D be a succinct cell-probe data structure for storing a walk $(v_1, \ldots, v_n)$ over general (directed) graphs G, using $\sum_i \lg(\deg(v_i)) + r$ bits of space, and query time t for each v_i. Then for any constant-size alphabet Σ, there is a succinct dictionary storing $x \in \Sigma^n$, with space $H_0(x) + r$ bits and query time query time t, where $H_0(x)$ is the* zeroth-order *empirical entropy of x. This reduction holds for any input string x with empirical frequencies which are inverses of powers of 2 lower bounded by a constant.*

[4] Once again, this lookup table is independent of the walk and only contains precomputed information about G for decoding, hence its "amortized" space cost is small as it is shared across instances.

[5] This seems a minor condition when dealing with *arbitrary* (nonuniform) priors μ, as the "hard" part is variable-length coding.

This reduction formalizes the intuition that the bottleneck in both problems is *variable-length* coding (unlike the case of regular graphs), and provides evidence that Theorem 2 might be optimal (Indeed, in the bit-probe model ($w = 1$), it is known that any succinct Dictionary with constant or even $r = O(\lg n)$ redundancy, must have $\Omega(\lg n)$ decoding time [Vio12]. For the related *succinct Partial Sums* problem [PV10], an $\Omega(\lg n)$ lower bound holds even in the cell-probe model ($w = \Theta(\lg n)$)).

Remark 1. *It is noteworthy that the assumption throughout the paper that the underlying graph G is of finite size is necessary: A length-n random walk on the undirected n-cycle is equivalent to the succinct* Partial Sums *problem, for which there is a cell-probe lower bound of $t \geq \Omega(\lg n / \lg \lg n)$ for constant redundancy [PV10]. A key feature that circumvents this lower bound in our proofs is the fact that the walk* mixes *fast (which doesn't happen on the n-cycle). This justifies the restriction to fixed sized graphs.*

1.2 Technical Overview

At a high level, our data structures use a *two-level* dictionary scheme to encode the random walk. Let us first focus on d-regular graphs G. To store a walk on G, we begin by storing the set of vertices V_i that are $\Theta(\lg n)$-far apart in the walk (called the *milestones*) using the succinct dictionary of [DPT10] (which we will refer to as *the* `DPT` *dictionary* in the following). The `DPT` dictionary is able to store any string $z \in \Sigma^k$ with only constant redundancy ($k \lg |\Sigma| + O(1)$ bits in total) and allow one to retrieve each z_i in constant time (Theorem 5). In particular, when the string z is drawn from the *uniform* distribution, its space usage matches the input entropy. When G is regular (and hence has a uniform stationary distribution), this is indeed the case, as a standard mixing-time argument implies that the milestones are very close to being *uniform and independent*.

The second important feature of milestones is that they break the dependence across the remaining vertices in the walk. That is, the milestones partitions the walk into "blocks" B_j of length $\Theta(\lg n)$. The Markov property of a random walk implies that these blocks of vertices are independent conditioned on the milestones. Thus, the next idea is to use another (separate) dictionary to encode the vector of intermediate blocks $(B_1, \ldots, B_{O(n/\lg n)}) \in w^{O(n/\lg n)}$, *conditioned* on the milestones. Again because of the mixing-time argument, for each block B_j, the number of possible subwalks in the block *given* the two milestones V_i and $V_{i+O(\lg n)}$ is always approximately $d^{|B_i|}/|G|$, *regardless of* the actual values that V_i and $V_{i+O(\lg n)}$ take. Hence, one can encode each subwalk using an integer no larger than $|\Sigma| = (1 + o(1))d^{|B_i|}/|G|$, and the second dictionary is used to succinctly store these integer encodings. When the input is uniformly random, these integer encodings are uniform and independent, hence `DPT` matches the entropy. Also, note that since each block B_j is of length $O(\lg n)$, it fits in a constant number of words (as $|G| = O(1)$). To see why a vertex between two milestones V_i and $V_{i+\Theta(\lg n)}$ can be retrieved efficiently, note that the `DPT` dictionary allows us to retrieve each symbol in the vector of milestones in constant time. Therefore, it suffices to retrieve V_i, $V_{i+\Theta(\lg n)}$ as well as the block B_j between them using a constant number of probes to both dictionaries. This gives us enough information to recover the entire subwalk from V_i to $V_{i+\Theta(\lg n)}$, and in particular, to recover the queried vertex. Although the above argument assumes a uniformly random input, we emphasize that our data structure works for *worst-case* input and queries.

Dealing with general graphs is a different ballgame, and the above construction does not obtain the claimed worst-case space bound. The main reason is that for a uniformly sampled random length-n walk, the marginal distribution of each (milestone) V_i may be arbitrary and non-uniform. Hence, we cannot apply the `DPT` dictionary directly on the milestone-vector, as for non-uniform vectors its space usage would be much higher than the input entropy. To overcome this issue, we use an idea inspired by rejection-sampling: We consider not only each milestone, but also the subwalks near it (with close indices). We partition the set of subwalks into *bundles*, such that for a uniformly random input, the bundle that contains the given subwalk is uniform. More precisely, for each milestone V_i,

1. a bundle is a subset of length-$2l$ subwalks that the input $(V_{i-l}, V_{i-l+1}, \ldots, V_{i+l})$ can take the value, for $l = \Theta(\lg n)$;

2. all subwalks in the same bundle have an *identical* middle vertex $V_i = v_i$ for some $v_i \in [|G|]$;
3. each bundle consists of approximately the same number of subwalks, i.e., for a uniformly random input, the bundle that contains it is roughly uniform;
4. for different V_i, the bundles are almost "independent."

We prove the existence of such good partition to subwalks using a spectral argument which helps control the number of subwalks delimited by any fixed pair of vertices (u, v). Given such bundling, instead of using DPT to store the milestones themselves, we use it to store the *name of the bundle* near each milestone. By Item 3 and 4 above, when the walk is uniformly random, the bundles to be stored are *uniform and independent*. Hence, the size of DPT dictionary matches the input entropy, as desired. The second part of the data structure is similar to the regular graph case at the high level. We store the blocks between the consecutive milestones, but now conditioned on *the bundles* (not the milestones). The query algorithm is also similar: to retrieve a vertex between two consecutive milestones, we first retrieve the bundles that contain the subwalks near them, and then retrieve the block, which is encoded conditioned on the bundles. The actual construction for the non-regular case is much more technical than regular graphs. See Section 3 for details.

The final challenge in our scheme is implementing the decoding process on a RAM – The above scheme only provides a *cell-probe* data structure (assuming arbitrary operations on $O(\lg n)$-bit words), since the aforementioned encoding of blocks + bundles is extremely implicit. Therefore, decoding with standard RAM operations appears to require storing giant lookup tables to allow efficient retrieval. Circumventing this intuition is one of our main contributions. It turns out that the basic problem that needs to be solved is the following: Given a walk $(v_0, \ldots, v_l)$ of length l, for $l = O(\lg n)$, from x to y ($v_0 = x$ and $v_l = y$), encode this walk using an integer between 1 and the number of such walks $(e_x{}^\top (A_G)^l e_y)$, such that given an index $i \in [0, l]$, one can decode v_i efficiently. Note that both two endpoints x and y are given, and do not need to be encoded. They would ultimately correspond to the milestones, which have already been stored elsewhere. Our encoding procedure is based on a B-way divide-and-conquer. As an example, when $B = 2$, we first recursively encode $(v_0, \ldots, v_{l/2})$ and $(v_{l/2}, \ldots, v_l)$, and then "merge" the halves. Given the encoding of the two halves, the final encoding of the entire walk is the *index in the lexicographic order* of the triple: I) $v_{l/2}$, II) encoding of $(v_0, \ldots, v_{l/2})$, and III) encoding of $(v_{l/2}, \ldots, v_l)$. To decode v_i, we first decode the entire triple by enumerate all possible values for $v_{l/2}$, and count how many length-l walks have this particular value for $v_{l/2}$ (these counts can be stored in the lookup table). Then we recurse into one of the two halves based on the value of i. This gives us an $O(\lg l) = O(\lg \lg n)$ time decoding algorithm. We can generalize this idea to larger B, by recursing on each of the $1/B$-fraction of the input. However, the decoding algorithm becomes more complicated, as we cannot afford to recover the entire $O(B)$-tuple. It turns out that to efficiently decode, one will need to do a *predecessor search* on a set of $\exp(B)$ integers with *increasing gaps*. This set depends only on the underlying graph G, therefore, we can store this predecessor search data structure in the lookup table, taking $\exp(B)$ space. Now, the recursion only has $\lg_B l = \frac{\lg \lg n}{\lg B}$ levels, each level still takes constant time, obtaining the claimed tradeoff.

It is worth noting that the DPT dictionary supports *appending* extra symbols in constant amortized time, hence it supports the (append-only) *online* version of the problem. Since our data structure consists of two instances of DPT, when the vertices in the random walk is given one at a time, our random-walk data structure can be build *online* as well, with amortized constant update time. The sizes of the two instances of DPT may increase over time, but their *ratio* remains fixed. By storing memory words of the two instances in *interleaving* memory locations with respect to the right ratio, we will not need to relocate the memory words when new vertices arrive.

For the harder task for matching the information-theoretic minimum space (i.e., the point-wise empirical entropy of the walk $\sum_i \lg(\deg(v_i))$), we show how to efficiently encode the random-walk problem (on arbitrary constant-size graphs) using Pătraşcu's *aB-trees* [Pat08]. This provides a constant-redundancy scheme but increases decoding time to $t = O(\lg n)$. We provide evidence that this blowup might indeed be necessary: We design a *succinct reduction* showing how to "embed" the classic *Dictionary* problem on inputs $x \in \mu^n$, as a random-walk on the *Huffman Tree* of μ, augmented with certain

directed paths to enable fast decoding (this works for any distribution μ where each $\mu(i)$ is the inverse of a power of 2). This reduction shows that any asymptotic improvement on the aforementioned random walk data structure would lead to a improvement on the long-standing dictionary problem (see Theorem 9).

1.3 Related work

The "prior-free" version of the (generic) LDSC problem has been studied in the pattern-matching and information-theory communities, where in this setting the compression benchmark is typically some (high-order) empirical entropy $H_k(x)$ capturing "bounded-correlations" in the text (see e.g. [Man01] for a formal definition). It has been shown that classical *universal compression* schemes (such as Lempel-Ziv [ZL78] compression as well as the Burrows-Wheeler transform [BW94]) can be made to have random-access to individual symbols at the price of small (but not *succinct*) loss in compression (e.g., [FM05, DLRR13, TBW18, SW]). In general, these results are incomparable to our distributional setting of the LDSC problem, as the space analysis is asymptotic in nature and based on the (ergodic) assumption that the input $(X_1, \ldots, X_n)$ has finite-range correlations or restricted structure (Indeed, this is confirmed by our impossibility result for general LDSC in Corollary 3). In the case of k-order Markov chains, where the k'th empirical and Shannon entropies actually match ($H_k(X) = H(X)$), all previously known LDSC schemes generally have redundancy at least $r \geq \Omega(n/\lg n)$ regardless of the entropy of the source and decoding time (see e.g., [DLRR13, TBW18] and references therein). In contrast, our schemes for achieve $r = O(1)$ redundancy and constant (or at most logarithmic) query time on a RAM.

Technically speaking, the most relevant literature to our paper is the work on *succinct Dictionary* problem [BMRV02, Pag02, RRS07, GRR08, Pat08, DPT10] (see Section 4.1 for the formal problem definition). The state-of-art, Due to Pătraşcu [Pat08], is a succinct data structure (LDSC) for *product distributions* on n-letter strings $\overline{X} \sim \mu^n$, with an *exponential* tradeoff between query time and redundancy $r = O(n/(\lg n/t)^t)$ over the expected (i.e., zeroth-order) entropy of $\overline{X}$. Whether this tradeoff is optimal is a long-standing open problem in succinct data structures (more on this in Section 4.1). Interestingly, if μ is the *uniform* distribution, a followup work of Dodis et. al [DPT10] showed that constant redundancy and decoding time is possible on a word-RAM. (see Theorem 5, which will also play a key role in our data structures). In some sense, our results show that such optimal tradeoff carries over to strings with finite-length correlations.

2 Preliminaries

2.1 Graphs and Random walks.

Let G be an unweighed graph, A be its adjacency matrix, and P be the transition matrix of the random walk on G. P is A with every row normalized to sum equal to 1.

Undirected regular graphs. When G is undirected and d-regular, A is real symmetric, and $P = \frac{1}{d}A$. All of the eigenvalues are real. In particular, the largest eigenvalue of A is equal to d, with eigenvector $\mathbf{1}$, the all-one vector. Moreover, if G is connected and non-bipartite, then all other eigenvalues have absolute values strictly less than d. Suppose all other eigenvalues are all at most $(1-\epsilon)d$, the following lemma is known.

Lemma 1. *Let X be a vector of dimension $|G|$ corresponding to some distribution over the vertices. Let U be the vector corresponding to the uniform distribution. We have*

$$\|\frac{1}{d} \cdot A(X-U)\|_2 \leq (1-\epsilon)\|X-U\|_2.$$

That is, every step of the random walk on G makes the distribution X close to the uniform by a constant factor.

Strongly connected aperiodic graphs. For directed graph G, it is strongly connected if every node can be reached from every other node via directed paths. For strongly connected G, it is aperiodic if the greatest common divisor of the lengths of all cycles in G is equal to 1. One may view strongly connectivity and aperiodicity as generalization of connectivity and non-bipartiteness for undirected graphs from above.

For strongly connected aperiodic G, the Perron–Frobenius theorem asserts that

- let λ be the spectral radius of A, then λ is an eigenvalue with multiplicity 1;
- let $\pi^\top$ and σ be its left and right eigenvectors with eigenvalue λ respectively, all coordinates of π and σ are positive.

Similarly, for the transition matrix P,

- it has spectral radius 1, and 1 is an eigenvalue with multiplicity 1;
- let $\nu^\top$ be its left eigenvector, then $\nu^\top$ is the unique stationary distribution.

Moreover, let X_1 and X_2 be two vectors, we have the following approximation on $X_1^\top A^l X_2$ (e.g., see [Fil91]).

Lemma 2. *We have*

$$X_1^\top A^l X_2 = \lambda^l \cdot \left(\frac{\langle \sigma, X_1 \rangle \langle \pi, X_2 \rangle}{\langle \sigma, \pi \rangle} \pm O(\|X_1\|_2 \|X_2\|_2) \cdot \exp(-\Omega(l)) \right).$$

Proof (sketch). Let us first decompose X_2 into two vectors $\alpha \cdot \sigma$ and X' such that $\langle \pi, X' \rangle = 0$. Thus, we have

$$\alpha = \frac{\langle \pi, X_2 \rangle}{\langle \sigma, \pi \rangle},$$

and

$$X' = X_2 - \alpha \cdot \sigma.$$

For the first term, we have $A^l(\alpha \cdot \sigma) = \lambda^l \alpha \cdot \sigma$. To estimate the second term, let $D = \mathrm{diag}(\sigma/\pi)$, i.e., D is the diagonal matrix with the i-th diagonal entry equal to σ_i/π_i. Since A is strongly connected and aperiodic, there exists a constant c such that all entries in A^c are positive. We will show that $\lambda^{-2l} \cdot X'^\top (A^\top)^l D^{-1} A^l X'$ decreases exponentially. To this end, consider the matrix

$$D^{1/2} (A^\top)^c D^{-1} A^c D^{1/2}.$$

This matrix is positive real symmetric, and $\pi^\top D^{1/2}$ is its left eigenvector with eigenvalue λ^{2c}. Since $\pi^\top D^{1/2}$ is positive, by the Perron–Frobenius theorem, all other eigenvalues have magnitude strictly less than λ^{2c}, and assume they are all at most $((1-\epsilon)\lambda)^{2c}$ for some $\epsilon > 0$.

For every i, since $\left\langle \pi^\top D^{1/2}, D^{-1/2} A^i X' \right\rangle = 0$, we have

$$X'^\top (A^\top)^{i+c} D^{-1} A^{i+c} X' \le ((1-\epsilon)\lambda)^{2c} \cdot X'^\top (A^\top)^i D^{-1} A^i X'.$$

Therefore, we have

$$\begin{aligned} X'^\top (A^\top)^l D^{-1} A^l X' &\le \exp(-\Omega(l)) \cdot \lambda^{2l} X'^\top D^{-1} X' \\ &\le \exp(-\Omega(l)) \cdot \lambda^{2l} \cdot O(\|X'\|_2^2) \\ &\le \exp(-\Omega(l)) \cdot \lambda^{2l} \cdot O(\|X_2\|_2^2). \end{aligned}$$

Thus, $\|A^l X'\|_2 \le \lambda^l \cdot O(\|X_2\|_2) \cdot \exp(-\Omega(l))$.

Combining the two parts, we have

$$X_1^\top A^l X_2 = \frac{\langle \sigma, X_1 \rangle \langle \pi, X_2 \rangle}{\langle \sigma, \pi \rangle} \cdot \lambda^l \pm \lambda^l \cdot O(\|X_1\|_2 \|X_2\|_2) \cdot \exp(-\Omega(l)).$$

This proves the lemma. □

One can also prove a similar statement about P.

Lemma 3. $X_1^\top P^l X_2 = \langle \mathbf{1}, X_1 \rangle \langle \nu, X_2 \rangle \pm O(\|X_1\|_2 \|X_2\|_2) \cdot \exp(-\Omega(l))$.

2.2 Space Benchmarks

There are $\mathbf{1}^\top A^n \mathbf{1}$ different walks $(v_0, \ldots, v_n)$ on G of length n. Thus, $\lg \mathbf{1}^\top A^n \mathbf{1}$ bits is the optimal *worst-case* space for storing a n-step random walk.

However, for non-regular graphs, some walk may appear with a higher probability than the others. By Lemma 3, the marginal of V_i quickly converges to ν. Therefore, we have

$$H(V_{i+1} \mid V_i) = \sum_x \nu_x \cdot \lg \deg(x) + \exp(-\Omega(i)).$$

By chain-rule and the Markov property of a random walk, we have

$$H(V_0, \ldots, V_n) = n \cdot \left(\sum_x \nu_x \cdot \lg \deg(x) \right) + O(1).$$

This is the optimal *expected* space any data structure can achieve.

Finally, note that the *point-wise* space benchmark defined in the introduction

$$\lg |G| + \sum_{i=0}^{n-1} \lg \deg(v_i)$$

implies almost optimal expected, since it assigns $\lceil \log 1/p \rceil$ bits to a walk that appears with probability p.

2.3 Word-RAM model and Succinct Data Structures

The following surprising (yet limited) result of [DPT10] will be used as a key subroutine in our data structures.

Theorem 5 (Succinct dictionary for *uniform* strings, [DPT10])**.** *For any $|\Sigma| \leq w$ (not necessarily a power of 2), there is a succinct Dictionary storing any string $x \in \Sigma^n$ using $\lceil n \lg |\Sigma| \rceil$* bits *of space, supporting constant-time retrieval of any x_i, on a word-RAM with word size $w = \Theta(\lg n)$, assuming pre-computed lookup tables of $O(\lg n)$ words. Moreover, the dictionary supports the online ("append-only") version with constant update and query times.*

We remark that the cost of the pre-computed lookup tables is negligible and they are shared across instances (they are essentially small code-books for decoding and hence do not depend on the input itself). This is a standard requirement for variable-length coding schemes.

3 Succinct Data Structures for Storing Random Walks

3.1 Warmup: d-Regular Graphs

Fix a d-regular graph G with k vertices, we assume that G is connected and non-bipartite. Denote its adjacency matrix by A. In the following, we show that a walk on G can be stored succinctly, and allow efficient decoding of each vertex.

Theorem 6. *Given a walk $(v_0, \ldots, v_n)$ in G, there is a succinct* cell-probe *data structure that uses at most $\lg_2 |G| + n \lg_2 d + 3$ bits of memory, and supports retrieving each v_q in constant time for $q = 0, \ldots, n$, assuming the word-size $w \geq \Omega(\lg n)$. Moreover, the data structure can be implemented on a word RAM using an extra r-bit lookup table, which depends only on G, supporting vertex retrievals in $O(\frac{\lg \lg n}{\lg \lg r})$ time, for any $r \geq \Omega(\lg^2 n)$. The constants in the query times depend on the size of G.*

Proof. The idea is to divide the walk into blocks of length l for $l = \Theta(\lg n)$, so that if we take every l-th node in a uniformly random walk (which we call the milestones), they look almost independent. We first store all these milestones using the DPT dictionary, which introduces no more than one bit of redundancy. Then note that conditioned on the milestones, the subwalk between any two adjacent milestones are also independent and uniform over some set, then we store the subwalks *conditioned on the milestones* using DPT again.

More formally, the top eigenvalue of A is equal to d. Suppose all other eigenvalues are at most $(1-\epsilon)d$, we set $m = \lfloor \frac{\epsilon}{2} \cdot n/\ln n \rfloor$, and $l = n/m \geq \frac{2}{\epsilon} \ln n$. We divide the walk into m blocks of length approximately l each. Let $a_i = \lfloor (i-1) \cdot l \rfloor$ for $1 \leq i \leq m+1$, be the $m+1$ milestones. Note that $a_1 = 0$ and $a_{m+1} = n$. The i-th block is from the i-th milestone v_{a_i} to the $(i+1)$-th $v_{a_{i+1}}$. Hence, the length of the block is in $(l-1, l+1)$.

The first part of the data structure stores all milestones, i.e., v_{a_i} for all $1 \leq i \leq m+1$ using DPT. This part uses $\lceil (m+1) \lg_2 |G| \rceil$ bits of memory.

Next, we store the subwalks between the adjacent milestones. By Lemma 1, after l steps of random walk from any vertex v_{a_i}, the ℓ_2 distance to the uniform distribution U is at most $(1-\epsilon_k)^l \leq n^{-2}$, hence the ℓ_∞ distance to U is also at most n^{-2}. In particular, the probability that the random walk ends at each vertex is upper bounded by $1/|G| + 1/n^2$. That is, given the milestones v_{a_i} and $v_{a_{i+1}}$, the number of possible subwalks in the i-th block (from v_{a_i} to $v_{a_{i+1}}$) is always at most

$$\left(\frac{1}{|G|} + \frac{1}{n^2}\right) \cdot d^{a_{i+1}-a_i},$$

regardless of the values of v_{a_i} and $v_{a_{i+1}}$.

Hence, the subwalk can be encoded using a positive integer between 1 and $\lfloor (1/|G| + 1/n^2) \cdot d^{a_{i+1}-a_i} \rfloor$. Note that since $l = O(\lg n)$, this integer has $O(w)$ bits. For cell-probe data structures, one can hardwire an arbitrary encoding for every possible pair of milestones and every possible length of the block. We will defer the implementation on RAM to the end of this subsection (Lemma 4), and let us focus on the main construction for now.

We obtain an integer for each subwalk between adjacent milestones. We again use DPT to store these integers. This part uses at most

$$\left\lceil \sum_{i=1}^{m} \lg_2 \left(\left(1/|G| + 1/n^2\right) \cdot d^{a_{i+1}-a_i} \right) \right\rceil$$

bits of memory.

Therefore, the number of bits we use in total is at most

$$\begin{aligned}
&((m+1)\lg_2 |G| + 1) + \left(\sum_{i=1}^{m} \lg_2 \left(\left(|G|^{-1} + n^{-2}\right) \cdot d^{a_{i+1}-a_i} \right) \right) + 1 \\
&= (m+1)\lg_2 |G| + \sum_{i=1}^{m} (a_{i+1} - a_i) \lg_2 d + m \lg_2 (|G|^{-1} + n^{-2}) + 2 \\
&= \lg_2 |G| + (a_{m+1} - a_1) \lg_2 d + m \lg_2 (1 + |G| \cdot n^{-2}) + 2 \\
&\leq \lg_2 |G| + n \lg_2 d + m|G| \cdot n^{-2} + 2 \\
&\leq \lg_2 |G| + n \lg_2 d + 3.
\end{aligned}$$

The query algorithm in the cell-probe model is straightforward. To retrieve v_q, we first compute the block it belongs to: $i = \lceil q/l \rceil$. Then we query the first part of the data structure to retrieve both v_{a_i} and $v_{a_{i+1}}$, and query the second part to retrieve the integer that encodes the subwalk between them conditioned on v_{a_i} and $v_{a_{i+1}}$. They together recover the whole subwalk, and in particular, v_q.

□

Decoding on the word-RAM. Denote by $\mathcal{N}_l(x,y)$, the number of different walks from x to y of length l, i.e., $\mathcal{N}_l(x,y) = \mathbf{e}_x^\top A^l \mathbf{e}_y$. In the following, we show that for every x, y and $l = O(\lg n)$, there is a way to encode such a walk using an integer in $[\mathcal{N}_l(x,y)]$, which allows fast decoding.

Lemma 4. *For $l = O(\lg n)$, given a length-l walk from x to y, one can encode it using an integer $K \in [\mathcal{N}_l(x,y)]$ such that with an extra lookup table of size r, depending only on the graph G, one can retrieve the q-th vertex in the walk in $O(\frac{\lg\lg n}{\lg\lg r})$ time, for any $r \geq \Omega(\lg^2 n)$.*

Proof. To better demonstrate how we implement this subroutine, we will first present a solution with a slower decoding time of $O(\lg \frac{\lg n}{\lg r}) = O(\lg\lg n - \lg\lg r)$, and for simplicity, we assume l is a power of two for now.

Given a length-l walk from x to y, the encoding procedure is based on divide-and-conquer. We first find the vertex z in the middle of the walk, and recursively encode the two length-$l/2$ walks from x to z and from z to y (given the endpoints). Suppose from the recursion, we obtained two integers $K_1 \in [\mathcal{N}_{l/2}(x,z)]$ and $K_2 \in [\mathcal{N}_{l/2}(z,y)]$, then the final encoding will be the index in the lexicographic order of the triple (z, K_1, K_2). That is, we encode the walk by the integer

$$K = \sum_{z'<z} \mathcal{N}_{l/2}(x,z')\mathcal{N}_{l/2}(z',y) + (K_1 - 1)\mathcal{N}_{l/2}(z,y) + K_2.$$

Hence, K is an integer between 1 and $\sum_z \mathcal{N}_{l/2}(x,z)\mathcal{N}_{l/2}(z,y) = \mathcal{N}_l(x,y)$.

We store the constants $\mathcal{N}_{l'}(x,y)$ for all $l' \in [1,l]$ and vertices x, y in the lookup table. The decoding procedure is straightforward. Given x, y and K, to decode the q-th vertex in the walk, we first recover the triple (z, K_1, K_2). To this end, we cycle through all z in the alphabetic order: If $K > \mathcal{N}_{l/2}(x,z)\mathcal{N}_{l/2}(z,y)$, subtract K by $\mathcal{N}_{l/2}(x,z)\mathcal{N}_{l/2}(z,y)$, and increment z; Otherwise, the current z is the correct middle vertex. Then K_1 and K_2 can be computed by $K_1 = \lfloor (K-1)/\mathcal{N}_{l/2}(z,y) \rfloor + 1$ and $K_2 = (K-1) \bmod \mathcal{N}_{l/2}(z,y) + 1$. Next,

- if $q = l/2$, z is the queried vertex, and return z;
- if $q < l/2$, recursively query the q-th vertex in the length-$l/2$ walk from x to z;
- if $q > l/2$, recursively query the $(q - l/2)$-th vertex in the walk from z to y.

This gives us a decoding algorithm running in $O(\lg\lg n)$ time. To obtain a faster decoding time of $t < \lg\lg n$, we could run the above recursion for t levels, and arrive at a subproblem asking to decode a length-$(l/2^t)$ walk from an integer no larger than $2^{O(l/2^t)}$. Instead of continuing the recursion, we simply store answers to all possible such decoding subproblems in a lookup table of size $r = k^2 \cdot (l/2^t) \cdot 2^{O(l/2^t)} = 2^{O(2^{-t}\lg n)}$. Hence, the decoding time is $t = O(\lg \frac{\lg n}{\lg r})$.

To obtain the claimed decoding time of $O(\frac{\lg\lg n}{\lg\lg r})$, the encoding algorithm will be based on a B-way divide-and-conquer for some parameter $B > 2$. Given a length-l walk from x to y, we first consider $B+1$ vertices $(x =) z_1, z_2, \ldots, z_B, z_{B+1} (= y)$ such that z_i is the $\lfloor (i-1)l/B \rfloor$-th vertex in the walk, and recursively encode the length-l/B walks from z_i to z_{i+1} for $i = 1, \ldots, B$, obtaining integers $K_1, \ldots, K_B$. Next, we encode the vertices $z_2, \ldots, z_B$ by an integer $Z \in [|G|^{B-1}]$. The final encoding is according to the lexicographic order of the tuple $(Z, K_1, \ldots, K_B)$. To decode a vertex between z_i and z_{i+1}, the algorithm will need to recover z_i, z_{i+1} and K_i in order to recurse. However, when B is large, we cannot afford to enumerate Z as before. Instead, we will use a trick from [Pat08], which allows us to recover each z_i in constant time, as well as K_i.

More specifically, for each tuple $(z_2, \ldots, z_B)$, consider the number of walks that go through them at the corresponding vertices

$$\prod_{i=1}^{B} \mathcal{N}_{\lfloor il/B \rfloor - \lfloor (i-1)l/B \rfloor}(z_i, z_{i+1}).$$

In the lookup table, we store all these tuples in the *sorted order by this number* (break ties arbitrarily), using $|G|^{B-1} \cdot (B-1)\lceil \lg |G| \rceil$ bits. The encoding Z of the tuple will be the index in this order. Therefore, to decode Z from x, y and K, it suffices to find the largest Z such that the total number of walks corresponding to a tuple $(z_2, \ldots, z_B)$ ranked prior to Z, is smaller than K. This is precisely a *predecessor search* data structure problem over a set of size $|G|^{B-1}$ with *monotone gaps*, where the set consists of for all Z, the total number of walks corresponding to a tuple ranked prior to Z, and the query is K. Because we sort the tuples by the number of corresponding walks, the numbers in the set have monotone gaps. It was observed in [Pat08] that the monotone gaps allow us to solve predecessor search with linear space and constant query time. Hence, we further store in the lookup table, this linear space predecessor search data structure, using $O(|G|^{B-1})$ words (the standard predecessor search problem only returns the number, however, it is not hard to have the data structure also return the index Z.) From the predecessor search data structure, we obtain Z, hence z_i and z_{i+1} from the first lookup table above, as well as K', the rank of the tuple $(K_1, \ldots, K_B)$ given Z. Thus, K_i can be computed by

$$K_i = \lfloor (K'-1) / \prod_{i'=i+1}^{B} \mathcal{N}_{\lfloor i'l/B \rfloor - \lfloor (i'-1)l/B \rfloor}(z_{i'}, z_{i'+1}) \rfloor \mod \mathcal{N}_{\lfloor il/B \rfloor - \lfloor (i-1)l/B \rfloor}(z_i, z_{i+1}) + 1.$$

It can be computed in constant time, once we also have stored in the lookup table, for all $(z_2, \ldots, z_B)$ and all i, the number

$$\prod_{i'=i+1}^{B} \mathcal{N}_{\lfloor i'l/B \rfloor - \lfloor (i'-1)l/B \rfloor}(z_{i'}, z_{i'+1}).$$

Finally, the recursion has $O(\log_B l) = O(\frac{\lg \lg n}{\lg B})$ levels, and each level takes constant time. We store in the lookup table for all x, y and $l' \le l$, the above tables and the predecessor search data structure using in total $r = O(|G|^{B-1} \cdot l^2) = 2^{O(B)} \cdot \lg^2 n$ bits. Thus, for $r \ge \lg^2 n$, the decoding time is $O(\frac{\lg \lg n}{\lg \lg r})$. This proves the lemma. □

3.2 General Graphs

In this subsection, we prove our theorem for storing a walk in a general directed graph with respect to worst-case space bound. Fix a strongly connected aperiodic directed graph G. Let A be its adjacency matrix. The total number of length-n walks in G is equal to $\mathbf{1}^\top A^n \mathbf{1}$.

Theorem 7. *Let G be a strongly connected aperiodic directed graph. Given a length-n walk $(v_0, \ldots, v_n)$ on G, one can construct a data structure that uses $\lg(\mathbf{1}^\top A^n \mathbf{1}) + 5$ bits of space. Then there is a query algorithm that given an index $q \in [0, n]$, retrieves v_q in $O(\frac{\lg \lg n}{\lg \lg r})$ time with access to the data structure and a lookup table of $r \ge \Omega(\lg^2 n)$ bits, where the lookup table only contains precomputed information about G, on a word RAM of word-size $w = \Theta(\lg n)$. The constants in the query times depend on the size of G.*

The previous solution fails for general graph G. The main reason is that if we take a uniformly random length-n walk, the marginal distribution of each milestone is not uniform, i.e., the entropy of each milestone is strictly less than $\lg |G|$. This implies that if we store them as before, using $\lg |G|$ bits per milestone, this part of the data structure already introduces a prohibitive amount of redundancy (constant bits per milestone). Another closely related issue in the non-regular case is that the distribution on walks between any two milestones is no longer uniform.

To circumvent this issue, we partition the set of subwalks near each milestone into subsets, which we refer to as the *bundles*. More specifically, let $l = \Theta(\lg n)$ be a parameter. For each milestone v_i, we partition the set of all possible subwalks from v_i to v_{i+l} into groups $\{g_{x,j}\}$, such that

1. all subwalks in $g_{x,j}$ have $v_i = x$;
2. the size $|g_{x,j}|$ is roughly the same for all x and j;

3. if we sample a uniformly random walk from $g_{x,j}$, the marginal distribution of v_{i+l} is roughly the same for all x and j.

The second property above implies that if the input walk is random, then the group $g_{x,j}$ that contains the subwalk is uniformly random. The third property implies that which group $g_{x,j}$ contains the subwalk almost does not reveal anything about the walk after v_{i+l}, as the distribution already "mixes" at v_{i+l}. We then partition the set of subwalks from v_{i-l} to v_i into groups $\{h_{x,j}\}$ with the similar properties. Finally, a bundle $\mathcal{B}_{x,j_1,j_2}$ will be the product set of h_{x,j_1} and g_{x,j_2}, i.e., it is obtained by concatenating one subwalk from h_{x,j_1} and one from g_{x,j_2} (they both have $v_i = x$, and thus, can be concatenated). The bundles constructed in this way have the following important properties:

1. all subwalks in each bundle go through the same vertex at the milestone;
2. each bundle consists of approximately the same number of subwalks;
3. adjacent bundles are almost "independent".

The data structure first stores for each milestone v_i, which bundle contains the subwalk near it, using `DPT` (Item 2 and 3 above guarantee that this part introduces less than one bit of redundancy). Next, the data structure stores the exact subwalk between each pair of adjacent milestones, given bundles that contain (part of) it. In the following, we elaborate on this idea, and present the details of the data structure construction. To prove the theorem, we will use the following property about strongly connected aperiodic graphs, which is a corollary of Lemma 2.

Lemma 5. *Let λ be the largest eigenvalue of A, $\pi^\top$ and σ be its left and right eigenvectors, i.e., $\pi^\top A = \lambda\pi^\top$ and $A\sigma = \lambda\sigma$. Then σ and π both have positive coordinates. For large $l \geq \Omega(\lg n)$, we have the following approximation on $X_1^\top A^l X_2$ for vectors X_1 and X_2:*

$$X_1^\top A^l X_2 = \lambda^l \cdot \left(\frac{\langle \sigma, X_1 \rangle \cdot \langle \pi, X_2 \rangle}{\langle \sigma, \pi \rangle} \pm O(n^{-2} \cdot \|X_1\|_2 \|X_2\|_2) \right).$$

Proof of Theorem 7. Let $m = \Theta(n/\lg n)$ be an integer, such that $l = n/2m$ is large enough for Lemma 5 and $\lambda^l \geq n^4$. Similar to the regular graph case, we divide the walk into m blocks of roughly equal length. For $i = 0, \ldots, m$, let $a_i = \lfloor 2il \rfloor$. The i-th block is from the a_{i-1}-th vertex in the walk to the a_i-th, and each a_i is a milestone. Let b_i be the midpoint between the milestones a_i and a_{i+1}, i.e., $b_i = \lfloor (2i+1)l \rfloor$.

Consider the set of all possible subwalks from b_{i-1} to b_i, we will partition this set into *bundles*, such that all subwalks in each bundle has the same v_{a_i} (different bundles may *not* necessarily have different vertex at a_i). To formally define the bundles, let us first consider the subwalk from v_{a_i} to v_{b_i}, which has length $b_i - a_i \approx l$. In total, there are $\mathbf{1}^\top A^{b_i - a_i} \mathbf{1}$ subwalks of length $b_i - a_i$. For any vertices x, y, $\mathbf{e}_x^\top A^{b_i - a_i} \mathbf{1}$ of them have $v_{a_i} = x$, and $\mathbf{e}_x^\top A^{b_i - a_i} \mathbf{e}_y$ have $v_{a_i} = x$ and $v_{b_i} = y$. Using the notation from Lemma 4, we have $\mathcal{N}_{b_i - a_i}(x, y) = \mathbf{e}_x^\top A^{b_i - a_i} \mathbf{e}_y$.

Now we partition the set of all possible subwalks from v_{a_i} to v_{b_i} such that $v_{a_i} = x$ into

$$s_x := \left\lfloor \frac{\mathbf{e}_x^\top A^{b_i - a_i} \mathbf{1}}{\mathbf{1}^\top A^{b_i - a_i} \mathbf{1}} \cdot n^2 \right\rfloor$$

groups $g_{x,1}, \ldots, g_{x,s_x}$. By Lemma 5, we have

$$\begin{aligned} s_x &= \frac{\langle \sigma, \mathbf{e}_x \rangle \langle \pi, \mathbf{1} \rangle \lambda^{b_i - a_i}}{\langle \sigma, \mathbf{1} \rangle \langle \pi, \mathbf{1} \rangle \lambda^{b_i - a_i}} \cdot n^2 \left(1 \pm O(n^{-2})\right) \\ &= \frac{\sigma_x}{\langle \sigma, \mathbf{1} \rangle} \cdot n^2 (1 \pm O(n^{-2})). \end{aligned} \tag{5}$$

For each vertex y, we ensure that the number of subwalks with $v_{b_i} = y$ in each $g_{x,j}$ for $j \in [s_x]$ is approximately the same. More specifically, we use Lemma 4 to encode the subwalk from v_{a_i} to v_{b_i}

using an integer $K_2^{(i)} \in [\mathcal{N}_{b_i-a_i}(v_{a_i}, v_{b_i})]$. This subwalk belongs to $g_{x,j}$ for $x = v_{a_i}$ and

$$j = \left\lfloor \frac{(K_2^{(i)} - 1) \cdot s_x}{\mathcal{N}_{b_i-a_i}(v_{a_i}, v_{b_i})} \right\rfloor + 1. \tag{6}$$

Therefore, by Lemma 5 and Equation (5) for every x, y, j, the number of subwalks from x to y in $g_{x,j}$ is at most

$$\begin{aligned} \frac{\mathcal{N}_{b_i-a_i}(x,y)}{s_x} + 1 &= \frac{\sigma_x \pi_y}{\langle \sigma, \pi \rangle} \cdot \lambda^{b_i-a_i} \cdot \frac{\langle \sigma, \mathbf{1} \rangle}{\sigma_x} \cdot n^{-2}(1 \pm O(n^{-2})) \\ &= \frac{\langle \sigma, \mathbf{1} \rangle \cdot \pi_y}{\langle \sigma, \pi \rangle} \cdot \lambda^{b_i-a_i} n^{-2}(1 \pm O(n^{-2})). \end{aligned} \tag{7}$$

Note that it is important that the $+1$ term is absorbed into the $O(n^{-2})$ term, since $\lambda^{b_i-a_i} = \Theta(\lambda^l) \geq \Omega(n^4)$.

Similarly, we also partition the set of subwalks from $v_{b_{i-1}}$ to v_{a_i} into groups $h_{x,1}, \ldots, h_{x,t_x}$, for

$$t_x := \left\lfloor \frac{\mathbf{1}^\top A^{a_i-b_{i-1}} \mathbf{e}_x}{\mathbf{1}^\top A^{a_i-b_{i-1}} \mathbf{1}} \cdot n^2 \right\rfloor = \frac{\pi_x}{\langle \pi, \mathbf{1} \rangle} \cdot n^2 (1 \pm O(n^{-2})). \tag{8}$$

Again, we use Lemma 4 to encode the subwalk from $v_{b_{i-1}}$ to v_{a_i} using an integer $K_1^{(i)} \in [\mathcal{N}_{a_i-b_{i-1}}(v_{b_{i-1}}, v_{a_i})]$. This subwalk belongs to $h_{x,j}$ for $x = v_{a_i}$ and

$$j = \left\lfloor \frac{(K_1^{(i)} - 1) \cdot t_x}{\mathcal{N}_{a_i-b_{i-1}}(v_{b_{i-1}}, v_{a_i})} \right\rfloor + 1. \tag{9}$$

The number of subwalks from y to x in $h_{x,j}$ is at most

$$\frac{\mathcal{N}_{a_i-b_{i-1}}(y,x)}{t_x} + 1 = \frac{\sigma_y \cdot \langle \pi, \mathbf{1} \rangle}{\langle \sigma, \pi \rangle} \cdot \lambda^{a_i-b_{i-1}} n^{-2}(1 \pm O(n^{-2})). \tag{10}$$

Now we describe how we partition the subwalks from $v_{b_{i-1}}$ to v_{b_i} into bundles. Each bundle has the form $h_{x,j_1} \times g_{x,j_2}$ for some vertex x and $j_1 \in [t_x], j_2 \in [s_x]$. That is, we define the bundle $\mathcal{B}_{x,j_1,j_2}$ as

$$\mathcal{B}_{x,j_1,j_2} := \{p_1 \circ p_2 : p_1 \in h_{x,j_1}, p_2 \in g_{x,j_2}\},$$

where $p_1 \circ p_2$ concatenates the two paths. It is not hard to verify that $\{\mathcal{B}_{x,j_1,j_2}\}$ is a partition, and the number of different bundles is

$$\sum_x s_x t_x = \frac{\langle \sigma, \pi \rangle}{\langle \sigma, \mathbf{1} \rangle \langle \pi, \mathbf{1} \rangle} \cdot n^4 (1 \pm O(n^{-2})) \tag{11}$$

by Equation (5) and (8). In particular, the index of the bundle (x, j_1, j_2) can be encoded using the integer

$$\sum_{x' < x} s_{x'} t_{x'} + (j-1) t_x + j_2.$$

As special cases, for $i = 0$, the subwalk is from v_{a_0} to v_{b_0}, and the bundles are defined to be $\mathcal{B}_{x,j_2} := g_{x,j_2}$ for vertex x and $j_2 \in [s_x]$. For $i = m$, the bundles are $\mathcal{B}_{x,j_1} := h_{x,j_1}$ for vertex x and $j_1 \in [t_x]$. In both cases, the number of bundles is at most n^2.

Data structure construction and space analysis. Now, we are ready to describe how to construct the data structure from the input $(v_0, \ldots, v_n)$. We first use Lemma 4 to encode the subwalks from v_{a_i} to v_{b_i} obtaining $K_2^{(i)}$, and subwalk from $v_{a_{i-1}}$ to v_{b_i} obtaining $K_1^{(i)}$, and compute the $m+1$ bundles consisting of each subwalk. More specifically, we compute $j_1^{(1)}, \ldots, j_1^{(m)}$ and $j_2^0, \ldots, j_2^{(m-1)}$ using Equation (9) and (6).

The first part of the data structure stores the indices of the $m+1$ bundles using DPT. These indices can be stored using

$$\left\lceil 2 \lg n^2 + (m-1) \lg \left(\sum_x s_x t_x \right) \right\rceil + 1$$

bits of space, which by Equation (11) is at most

$$4 \lg n + (m-1) \lg \frac{\langle \sigma, \pi \rangle \cdot n^4}{\langle \sigma, \mathbf{1} \rangle \langle \pi, \mathbf{1} \rangle} + O(m \cdot n^{-2}) + 2. \tag{12}$$

Next, we compute the index of the subwalk *within* each bundle. More specifically, we compute $k_1^{(1)}, \ldots, k_1^{(m)}$ and $k_2^0, \ldots, k_2^{(m-1)}$ as follows:

$$k_1^{(i)} = K_1^{(i)} - \left\lceil (j_1^{(i)} - 1) \cdot \frac{\mathcal{N}_{a_i - b_{i-1}}(v_{b_{i-1}}, v_{a_i})}{s_x} \right\rceil,$$

and

$$k_2^{(i)} = K_2^{(i)} - \left\lceil (j_2^{(i)} - 1) \cdot \frac{\mathcal{N}_{b_i - a_i}(v_{a_i}, v_{b_i})}{t_x} \right\rceil.$$

They are the indices *within* each $g_{x,j}$ and $h_{x,j}$.

The second part of the data structure stores the subwalk between the milestones within the bundles. In particular, we store the triples $(v_{b_i}, k_1^{(i)}, k_2^{(i)})$ for every $i = 0, \ldots, m-1$. By Equation (7) and (10), the number of triples is at most

$$\begin{aligned}
&\sum_y \frac{\langle \sigma, \mathbf{1} \rangle \pi_y}{\langle \sigma, \pi \rangle} \cdot \lambda^{b_i - a_i} n^{-2} \cdot \frac{\sigma_y \langle \pi, \mathbf{1} \rangle}{\langle \sigma, \pi \rangle} \cdot \lambda^{a_{i+1} - b_i} \cdot n^{-2} \cdot (1 \pm O(n^{-2})) \\
&= \frac{\langle \sigma, \mathbf{1} \rangle \langle \pi, \mathbf{1} \rangle}{\langle \sigma, \pi \rangle} \cdot \lambda^{a_{i+1} - a_i} n^{-4} (1 \pm O(n^{-2})).
\end{aligned}$$

Again, we store these triples using DPT. The total space of the second part is at most

$$m \lg \frac{\langle \sigma, \mathbf{1} \rangle \langle \pi, \mathbf{1} \rangle}{\langle \sigma, \pi \rangle n^4} + n \lg \lambda + O(m \cdot n^2) + 2. \tag{13}$$

Finally, summing up Equation (12) and (13), the total space usage of the data structure is at most

$$\begin{aligned}
&4 \lg n + (m-1) \lg \frac{\langle \sigma, \pi \rangle \cdot n^4}{\langle \sigma, \mathbf{1} \rangle \langle \pi, \mathbf{1} \rangle} + m \lg \frac{\langle \sigma, \mathbf{1} \rangle \langle \pi, \mathbf{1} \rangle}{\langle \sigma, \pi \rangle n^4} + n \lg \lambda + O(m \cdot n^{-2}) + 4 \\
&\leq \lg \frac{\langle \sigma, \mathbf{1} \rangle \langle \pi, \mathbf{1} \rangle}{\langle \sigma, \pi \rangle} + n \lg \lambda + O(m \cdot n^{-2}) + 4 \\
&= \lg \left(\frac{\langle \sigma, \mathbf{1} \rangle \langle \pi, \mathbf{1} \rangle}{\langle \sigma, \pi \rangle} \cdot \lambda^n \right) + O(m \cdot n^{-2}) + 4
\end{aligned}$$

which by Lemma 5 again, is at most

$$\leq \lg(\mathbf{1}^\top A^n \mathbf{1}) + 5.$$

Query algorithm. Given an integer $q \in [0, n]$, we first compute the block that contains v_t, suppose $a_i \le q \le a_{i+1}$. Suppose q is in the first half ($a_i \le q \le b_i$), we use the query algorithm of DPT on the first part of the data structure, retrieving the index of the bundle containing the subwalk from b_{i-1} to b_i, $(v_{a_i}, j_1^{(i)}, j_2^{(i)})$. Similarly, we retrieve the triple $(v_{b_i}, k_1^{(i)}, k_2^{(i)})$ from the second part. Then the encoding of the subwalk from v_{a_i} to v_{b_i} can be computed

$$K_2^{(i)} = k_2^{(i)} + \left\lceil (j_2^{(i)} - 1) \cdot \frac{\mathcal{N}_{b_i - a_i}(v_{a_i}, v_{b_i})}{t_x} \right\rceil.$$

Finally, we use the query algorithm of Lemma 4 to decode the $(q - a_i)$-th vertex in this subwalk, which is v_t. The case where q is in the second of the block ($b_i \le q \le a_{i+1}$) can be handled similarly. With an extra lookup table of size r for Lemma 4, each query can be answered in $O(\frac{\lg \lg n}{\lg \lg r})$ time. This proves the theorem. □

The above data structure also extends to general directed graphs.

Corollary 1. *With an extra $O(1)$ bits of space, Theorem 7 also applies to general strongly connected graph G.*

Proof. Suppose G is periodic with period p. Then its vertices can be divided into p sets $V_1, \ldots, V_p$ such that each vertex V_i only has outgoing edges to V_{i+1} (defining $V_{p+1} = V_1$). We can reduce the problem to the aperiodic case as follows.

Let us first only consider walks that both start and end with vertices in V_1. Therefore, the length of the walk n must be a multiple of p. We divide the walk into subwalks of length p, and view each length-p subwalk as a vertex in a new graph. That is, consider the graph $G^p|_{V_1}$ whose vertices are all possible length-p walks that start and end with vertices in V_1 in G, such that directed edges connect two subwalks if they can be concatenated, i.e., if the last vertex in u is the same as the first vertex in u', then there is an edge from u to u'. It is easy to verify that this graph is strongly connected and aperiodic, and a length-n walk on G can be viewed as a length-n/p walk on this graph. Then we apply Theorem 7 to store the length-n/p walk on $G^p|_{V_1}$. To retrieve v_q, it suffices to retrieve the $\lfloor q/p \rfloor$-th vertex in the walk on $G^p|_{V_1}$, and recover v_q via a look-up table.

For general inputs that do not necessarily start or end in V_1, we store a prefix and suffix of walk of length at most p naively, such that the remaining middle part starts and ends in V_1, for which we use the above construction. □

Finally, our data structure also applies to general directed graphs.

Corollary 2. *With an extra $O(\lg n)$ bits of space, Theorem 7 also applies to general directed graph G.*

Proof. We precompute the strongly connected components (SCC) of G. Given input walk $(v_0, \ldots, v_n)$, observe that it can only switch between different SCCs a constant number of times, since after the walk leaves an SCC, it could never come back. Hence, we first store the steps in the input that switch from one SCC to another using $O(\lg n)$ bits, then apply Corollary 1 on each subwalk within an SCC. □

3.3 Matching the point-wise optimal space for non-regular graphs

The data structure from Theorem 7 used approximately $\lg \mathbf{1}^\top A_G \mathbf{1}$ bits of space uniformly on *any* possible input sequence. This means that our data structure achieves the best possible space for a *uniformly chosen* length-n walk in G, by Shannon's source coding theorem. However, the distribution we actually care about is that of a random walk of length n starting from a random vertex in G, which for *non-regular* graphs may have much lower Shannon entropy. In the other words, for non-regular G, the following two distributions may be very far in KL divergence:

- ν_n: A uniformly chosen length-n walk in G.

- μ_n: An n-step random walk starting from a random vertex in G.

By Huffman coding, $H(\mu_n)$ is therefore the correct *expected* space benchmark. To achieve this expectation, the corresponding *poin-wise* space benchmark per walk $(v_0, \ldots, v_n) \sim \mu_n$ in G is

$$\lceil \lg |G| + \sum_{i=0}^{n-1} \lg \deg(v_i) \rceil$$

bits of space, since the space allocated to each sequence $(v_0, \ldots, v_n)$ is approximately $1/\lg \Pr[(v_0, \ldots, v_n)]$.

To match the above point-wise space bound, we use the *augmented B-tree* from [Pat08]. Fix a parameter $B \geq 2$, [Pat08] defines data structure *aB-trees* as follows:

- The data structure stores an array $A \in \Sigma^n$. The data structure is a B-ary tree with n leaves storing elements in A.
- Every node is augmented with a label from some alphabet Φ, such that the label of the i-th leaf is a function of $A[i]$, and the label of an internal node is a function of the labels of its B children, and the size of the subtree.
- The query algorithm examines the label of the root's children, decides which child to recurse on, examines all labels of that child's children, recurses to one of them, and so on. The algorithm must output the query answer when it reaches a leave.

Pătraşcu proves a general theorem to compress *any* such aB-tree, almost down to its input entropy:

Theorem 8 ([Pat08], Theorem 8)**.** *Let $B \leq O(\frac{w}{\lg(n+|\Phi|)})$, and let $\mathcal{N}(n, \varphi)$ be the number of instances of $A \in \Sigma^n$ that has the root labeled by φ. An aB-tree of size n with root label φ can be stored using $\lg_2 \mathcal{N}(n, \varphi) + 2$ bits. The query time is $O(\lg_B n)$, assuming a precomputed lookup tables of $O(|\Sigma| + |\Phi|^{B+1} + B \cdot |\Phi|^B)$ words, which only depend on n, B and the aB-tree algorithm.*

We build an aB-tree on top of the input. The alphabet size $|\Sigma| = |G|$. For each node of the tree that corresponds to a subwalk $(v_l, \ldots, v_r)$, we set its label φ to be the triple (v_l, v_r, S), such that S is an integer equal to

$$\sum_{i=l}^{r-1} \lceil n \lg_2 \deg(v_i) \rceil,$$

with the exception of root, whose label is only the integer $S = \sum_{i=0}^{n-1} \lceil n \lg_2 \deg(v_i) \rceil$, without v_0 and v_n. In the other words, S encodes the optimal "space" that this subsequence uses. Thus, the alphabet size of the labels $|\Phi|$ is $O(n^2)$. The label of a leaf v_l is $(v_l, v_l, 0)$, a function of v_l. To see why the label of an internal node is a function of the labels of its B children, suppose the B children have labels (v_l, v_{m_1-1}, S_1), $(v_{m_1}, v_{m_2-1}, S_2)$, …, $(v_{m_{B-1}}, v_r, S_B)$ respectively. Then the label of this node is (v_l, v_r, S) (or just S for the root) for

$$S = \sum_{i=1}^{B} S_B + \sum_{i=1}^{B-1} \lceil n^2 \lg_2 \deg(v_{m_i-1}) \rceil.$$

Theorem 8 compresses this data structure to $\lg_2 \mathcal{N}(n+1, \varphi) + 2$ bits, where ϕ is the label of the root, and supports queries in $O(\lg_B n)$ time. The following lemma bounds the value of $\mathcal{N}(n+1, \varphi)$.

Lemma 6. *For root label $\varphi = S$, we have $\mathcal{N}(n+1, \varphi) \leq 2^{\lg |G| + S/n}$.*

Proof. Let V be a uniformly random walk $V = (V_0, \ldots, V_n)$ with root label φ. Then $H(V) = \lg_2 \mathcal{N}(n+1, \varphi)$. On the other hand, by the chain rule and the fact that conditioning only reduces entropy, we have

$$H(V) = H(V_0, \ldots, V_n) \leq H(V_0) + \sum_{i=0}^{n-1} H(V_{i+1}|V_i) \leq \lg |G| + \sum_{i=0}^{n-1} \mathbb{E}[\lg \deg(v_i)] \leq \lg |G| + S/n,$$

where the last inequality is by definition of $\mathcal{N}(n, \varphi)$. Rearranging sides completes the proof. □

Therefore, the space usage is at most

$$\lg |G| + S/n + 2 \leq \lg |G| + \sum_{i=0}^{n-1} \lg \deg(v_i) + 3.$$

In particular, setting $B = 2$ gives us query time $O(\lg n)$ and lookup table size $\tilde{O}(n^6)$.

4 Lower Bounds

4.1 A Succinct Reduction from Dictionaries to Directed Random Walks

In this section, we exhibit a succinct reduction from the well-studied Dictionary data structure problem to the LDSC problem of storing random walks on (directed) non-regular graphs. Before we describe the reduction, we begin with several definitions. Recall that the zeroth-order *empirical entropy* of a string $x \in \Sigma^n$ is $H_0(x) := \sum_{\sigma \in \Sigma} f_\sigma \lg(n/f_\sigma(x))$ where f_σ is the number of occurrences (frequency) of the symbol σ in x. The *succinct Dictionary* problem is defined as follows:

Definition 1 (Succinct Dictionaries)**.** *Preprocess an n-letter string $x \in \Sigma^n$ into $H_0(x) + r$ bits of space, such that each x_i can be retrieved in time t. We denote this problem* $\mathsf{Dictionary}_{\Sigma,n}$.

Note that in the special case of bit-strings ($\Sigma = \{0, 1\}$), this is the classic *Membership* problem [Pag02, GM07, Pat08]. The best known time-space tradeoff for $\mathsf{Dictionary}_{\Sigma,n}$ (for constant-size alphabets) is $r = O(n/(\frac{\lg n}{t})^t)$ [Pat08].[6] While this exponential tradeoff is known to be optimal in the bit-probe model ($w = 1$) [Vio12], no cell-probe lower bounds are known, and this is one of the long-standing open problems in the field of succinct data structures.

A key component in our reduction will be the *Huffman tree* (c.f, Huffman code) of a distribution.

Definition 2 (Huffman Tree)**.** *The* Huffman Tree $\mathcal{T}_\mu$ *of a discrete distribution μ supported on alphabet Σ, is a rooted binary tree (not necessarily complete) with Σ leaves, one per alphabet symbol. For every symbol $\sigma \in \Sigma$, the unique root-to-leaf path to the symbol σ is of length exactly $\ell_\sigma := \lceil \lg_2(1/\mu(\sigma)) \rceil$.*

We remark that the existence of such trees for any discrete distribution μ is guaranteed by Kraft's inequality (for more context and construction of the Huffman tree, see [CT06]). We are now ready to prove Theorem 4, which we restate below.

Theorem 9. *Let D be a succinct cell-probe data structure for storing a walk $(v_1, \ldots, v_n)$ over general (directed) graphs G, using $\sum_i \lg(deg(v_i)) + r$ bits of space, and query time t for each v_i. Then for any constant-size alphabet Σ, there is a succinct dictionary storing $x \in \Sigma^n$, with space $H_0(x) + r$ bits and query time query time $t + 1$. This reduction holds for any input x with empirical frequencies $(f_\sigma(x)/n)$ which are inverses of powers of 2 lower bounded by a constant.*

Proof. First, we claim that the Dictionary problem can be reformulated as follows: Store an n-letter string drawn from some arbitrary *product* distribution $X \sim \mu^n$, using expected space $s = nH(\mu) + r$ bits, decoding X_i in time t, where $H(\mu)$ is the Shannon entropy of μ. Indeed, let $\mu = \mu(x)$ denote the empirical distribution of the input string $x \in \Sigma^n$ to $\mathsf{Dictionary}_{\Sigma,n}$. Since the zeroth-order space benchmark $H_0(x)$ only depends on the *marginal* frequencies of symbols in the input string, we may assume that each coordinate X_i is drawn *independently* from the induced distribution μ (which is arbitrary nonuniform), and $H_0(x) = H(\mu^n) = nH(\mu)$ holds.

[6] On a RAM, there is an extra $n^{1-\varepsilon}$ additive term for storing lookup-tables whose "amortized" cost is small, see Theorem 1 in [Pat08].

By this reformulation, we may assume the input to $\mathsf{Dictionary}_{\Sigma,n}$ is $X \sim \mu^n$ for some μ. We now define a (directed) graph such that a uniform random walk on this graph encodes the input string $X_1, \ldots, X_n \sim \mu^n$ *without losing entropy*. Let $\mathcal{T}_\mu$ be the Huffman Tree of the distribution μ, where all edges are directed downwards from root to leaves. By Definition 2, the probability that a random walk W on $\mathcal{T}_\mu$ starting from the root reaches leaf $\sigma \in \Sigma$ is precisely $2^{-\ell_\sigma} = 2^{-\lceil \lg_2(1/\mu(\sigma)) \rceil}$. If all probabilities in μ are (inverses of) powers of 2, then the probability of this event is exactly

$$\Pr[W \text{reaches leaf } \sigma] = 2^{-\lg_2(1/\mu(\sigma))} = \mu(\sigma). \tag{14}$$

In other words, under the assumption that all probabilities are powers of 2, sampling $X_i \sim \mu$ is equivalent to sampling a random leaf of $\mathcal{T}_\mu$ according to a uniform random walk starting from the root v_r. Let

$$d_\mu := \max_\sigma \ell_\sigma$$

be the height of the tree $\mathcal{T}_\mu$ (i.e., the length of the deepest path $\max_\sigma \lg(1/\mu(\sigma))$). To complete the construction of the graph, we connect each leaf (corresponding to symbol) σ back to the root of $\mathcal{T}_\mu$ via a vertex-disjoint *directed path* of length $d_\mu + 1 - \ell_\sigma$. Call the resulting graph G_μ. This simple trick makes sure that all "cycles" have *fixed* length $d_\mu + 1$, hence each sampled leaf in the walk can be decoded from a fixed block, despite the fact that each leaf is sampled at unpredictable depth. The key observation is that adding these vertex-disjoint paths does *not* increase the entropy of a random walk ($\lg_2(1) = 0$), but merely increases the size of the graph by a constant factor $|G_\mu| = O(|\Sigma| d_\mu) = O(1)$.

Let $(V_1, \ldots, V_{n'})$ be a uniform random walk of length $n' := (d_\mu + 1)n$ on G_μ starting from the root. By (14) and definition of G_μ, it follows that the unique leaf $V_{j_i} \in \Sigma$ sampled in the ith "cycle" of the walk ($j_i \in [(i-1)(d_\mu+1), i(d_\mu+1)]$), is distributed precisely as $X_i \sim \mu$. In particular, $H(V_{j_1}, \ldots, V_{j_n}) = nH(\mu)$, while the (conditional) entropies of all the rest of the V_j's are identically 0 (as all previous vertices $V_{<j_i}$ in the ith "cycle" are determined by V_{j_i} by the tree structure, and all remaining vertices $V_{>j_i}$ in the ith cycle correspond to the part of the walk that is a vertex-disjoint *path* and hence the conditional entropy of this subwalk is 0). Thus, by the chain rule and the premise of the theorem, we conclude that there is a succinct data structure storing a random walk $V_1, \ldots, V_n$ on G_μ using space $nH(\mu) + r$ bits of space, which by the above reformulation implies the same space for storing $x \in \Sigma^n$ up to the $O(|\Sigma| \lg n)$ additive term, so long as marginal frequencies $(f_\sigma(x)/n)$ are inverses of powers of 2.

Decoding of x_i is straightforward: Go to the second-before-last vertex $V_{i(d_\mu+1)-1}$ in the ith cycle (i.e., the one just before coming back to the root); Either $V_{i(d_\mu+1)-1}$ is a leaf of $\mathcal{T}_\mu$, or it belongs to a *unique* path corresponding to some leaf ℓ_σ (as all back-tracking paths are vertex disjoint) hence we know immediately that $X_i = \sigma$. By the premise of the theorem, the decoding time of each vertex in the walk is t, hence so is that of X_i. We conclude that the resulting succinct data structure is an (r, t)-Dictionary, as claimed.

□

4.2 Completeness of LDSC in the static cell-probe model

In this section, we prove unconditional (cell-probe) lower bounds on the LDSC problem, demonstrating that for some (in fact, most) classes of non-product distributions, this storage problem does not admit efficient time-space tradeoffs. Our first observation is that the LDSC problem under an *arbitrary* joint distribution μ is a "complete" static data structure problem:

Proposition 1 (Completeness of LDSC)**.** *For any prior μ on n files, $\mathsf{LDSC}^\mu_{n,\Sigma}$ is equivalent to some static data structure problem $\mathcal{P} = \mathcal{P}(\mu)$ with $|\mathcal{Q}| = n$ queries, on an input database Y of size $\mathbb{E}[|Y|] \leq H_\mu + 1$ bits. Conversely, any static data structure problem $\mathcal{P}$ with $|\mathcal{Q}|$ queries on a database $x \in \Sigma^n$, can be embedded as an* LDSC *problem on some $\mu = \mu(\mathcal{P})$, where $H_\mu(X) = n$. These equivalences hold in the cell-probe model with word size $w \geq \Omega(\lg |\Sigma|)$.*

Proof. The converse statement is straightforward: Given any static problem $\mathcal{P}$ on input $x \in \Sigma^N$ with query set $\mathcal{Q}$, consider the distribution $\mu(\mathcal{P})$ of $(q_1(X), q_2(X), \ldots, q_{|\mathcal{Q}|}(X))$, i.e., of all query answers $(A_1, \ldots, A_{|\mathcal{Q}|})$ to problem $\mathcal{P}$ under a uniformly random input database $X \in_R \Sigma^n$. The joint entropy of $\mu(\mathcal{P})$ is $H(A_1, \ldots, A_{|\mathcal{Q}|}) \leq H(X) = N \lg \Sigma$, since X determines all query answers. Hence this is a valid LDSC instance on $n = |\mathcal{Q}|$ files, over a joint distribution $\mu(\mathcal{P})$ with entropy $H_\mu \leq N \lg \Sigma \leq n$.

For the first direction of the proposition, let $\overline{X} := (X_1, \ldots, X_n)$ be the (random variable) input files to $\mathsf{LDSC}^{\mu}_{n,\Sigma}$, and let $Y := \mathsf{Huff}(X_1, \ldots, X_n)$ denote the *Huffman code* of the random variable $\overline{X}$. By the properties of the Huffman code (c.f. Definition 2), $Y = \mathsf{Huff}(\overline{X})$ is *invertible* and has expected length $\mathbb{E}_\mu[Y] \leq H_\mu(\overline{X}) + 1$ bits. This implies that Y is a uniformly distributed random variable with entropy $H(Y) = H(\overline{X})$. Now, define the data structure problem $\mathcal{P}$, in which the answer to the ith query is $\mathcal{P}(i, Y) := (\mathsf{Huff}^{-1}(Y))_i = X_i$ where the last inequality follows from the fact that Huffman coding is lossless and hence $\overline{X}$ is determined by *some* deterministic function $g(Y)$. While this provides a bound only on the expected size of the input ("database") Y, a standard Markov argument can be applied for bounding the worst-case size, if one is willing to tolerate an arbitrarily small failure probability of the data structure over the distribution μ.

□

Corollary 3 (LDSC Lower Bounds)**.** *Proposition 1 has the following lower bound implications:*

1. *(k-wise independent distributions) Using the cell-sampling arguments of [Sie04, Lar12], the above reduction implies that any linear-space data-structure (storing $O(H_\mu)$ bits) for $\mathsf{LDSC}^{\mu}{}_{n,\Sigma}$ when μ is an (H_μ)-wise independent distribution, requires*

$$t \geq \Omega(\lg H_\mu) = \Omega(\lg n)$$

   decoding time even in the cell-probe model (note that this type of tradeoff is the highest explicit lower bound known for any static problem).

2. *A rather simple counting argument ([Mil93]) implies that for* most *joint distributions μ with entropy $H(\mu) := h \ll n$, locally decodable source coding is impossible, in the sense that decoding requires $h^{1-o(1)}$ time unless trivial $\approx n^{1-o(1)}$ space is used (and this is tight by Huffman coding). Such implicit hard distribution μ can be defined by n random functions on a random h-bit string, where $h = n^{\varepsilon}$.*

References

[APW+08] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference*, ATC'08, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[BMRV02] H. Buhrman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? *SIAM J. Comput.*, 31(6):1723–1744, June 2002.

[BN13] Jérémy Barbay and Gonzalo Navarro. On compressing permutations and adaptive sorting. *Theor. Comput. Sci.*, 513:109–123, November 2013.

[BW94] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

[CS00] Tarun Chordia and Bhaskaran Swaminathan. Trading volume and cross-autocorrelations in stock returns. *Journal of Finance*, 55(2):913–935, 2000.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006.

[DLRR13] Akashnil Dutta, Reut Levi, Dana Ron, and Ronitt Rubinfeld. A simple online competitive adaptation of lempel-ziv compression with efficient random access support. In *2013 Data Compression Conference, DCC 2013, Snowbird, UT, USA, March 20-22, 2013*, pages 113–122, 2013.

[DPT10] Yevgeniy Dodis, Mihai Patrascu, and Mikkel Thorup. Changing base without losing space. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 593–602, 2010.

[Fil91] James Allen Fill. Eigenvalue bounds on convergence to stationarity for nonreversible markov chains, with an application to the exclusion process. *Ann. Appl. Probab.*, 1(1):62–87, 02 1991.

[FM05] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, July 2005.

[GM07] Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. *Theoretical Computer Science*, 379:405–417, July 2007.

[GPL+00] P Gopikrishnan, V Plerou, Y Liu, L.A.n Amaral, Xavier Gabaix, and H.e Stanley. Scaling and correlation in financial time series. *Physica A: Statistical Mechanics and its Applications*, 287(3):362–373, 2000.

[GRR08] Alexander Golynski, Rajeev Raman, and S. Srinivasa Rao. On the redundancy of succinct data structures. In *Algorithm Theory - SWAT 2008, 11th Scandinavian Workshop on Algorithm Theory, Gothenburg, Sweden, July 2-4, 2008, Proceedings*, pages 148–159, 2008.

[GY07] Mark Gales and Steve Young. The application of hidden markov models in speech recognition. *Found. Trends Signal Process.*, 1(3):195–304, January 2007.

[HBB+18] Kim M. Hazelwood, Sarah Bird, David M. Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. Applied machine learning at facebook: A datacenter infrastructure perspective. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*, pages 620–629, 2018.

[Lar12] Kasper Green Larsen. The cell probe complexity of dynamic range counting. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 85–94, 2012.

[Man01] Giovanni Manzini. An analysis of the burrows-wheeler transform. *J. ACM*, 48(3):407–430, May 2001.

[MCW15] Arya Mazumdar, Venkat Chandar, and Gregory W. Wornell. Local recovery in data compression for general sources. In *Proceedings - 2015 IEEE International Symposium on Information Theory, ISIT 2015*, volume 2015-June, pages 2984–2988, United States, 9 2015. Institute of Electrical and Electronics Engineers Inc.

[MHMP15] Ali Makhdoumi, Shao-Lun Huang, Muriel Médard, and Yury Polyanskiy. On locally decodable source coding. In *2015 IEEE International Conference on Communications, ICC 2015, London, United Kingdom, June 8-12, 2015*, pages 4394–4399, 2015.

[Mil93] Peter Bro Miltersen. The bit probe complexity measure revisited. In *STACS 93, 10th Annual Symposium on Theoretical Aspects of Computer Science, Würzburg, Germany, February 25-27, 1993, Proceedings*, pages 662–671, 1993.

[Pag02] Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM J. Comput.*, 31(2):353–363, February 2002.

[Pat08] Mihai Patrascu. Succincter. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 305–313, 2008.

[PV10] Mihai Pătraşcu and Emanuele Viola. Cell-probe lower bounds for succinct partial sums. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 117–122, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[RRS07] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4), November 2007.

[Sie04] Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM J. Comput.*, 33(3):505–543, 2004.

[SW] Sandip Sinha and Omri Weinstein. Local decodability of the burrows-wheeler transform. *Submitted, 2018*.

[TBW18] Kedar Tatwawadi, Shirin Saeedi Bidokhti, and Tsachy Weissman. On universal compression with constant random access. In *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*, pages 891–895, 2018.

[THOW16] Kedar Tatwawadi, Mikel Hernaez, Idoia Ochoa, and Tsachy Weissman. GTRAC: fast retrieval from compressed collections of genomic variants. *Bioinformatics*, 32(17):479–486, 2016.

[Vio12] Emanuele Viola. Bit-probe lower bounds for succinct data structures. *SIAM J. Comput.*, 41(6):1593–1604, 2012.

[WSY+16] Hao Wu, Xiaoyan Sun, Jingyu Yang, Wenjun Zeng, and Feng Wu. Lossless compression of jpeg coded photo collections. *Trans. Img. Proc.*, 25(6):2684–2696, June 2016.

[ZL78] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, September 1978.