Succinct Preferential-Attachment Graphs

Ziad Ismaili Alaoui* Namrata* Sebastian Wild*†

June 26, 2025

Abstract

Computing over compressed data combines the space saving of data compression with efficient support for queries directly on the compressed representation. Such data structures are widely applied in text indexing and have been successfully generalised to trees. For graphs, support for computing over compressed data remains patchy; typical results in the area of succinct data structures are restricted to a specific class of graphs and use the same, worst-case amount of space for any graph from this class.

In this work, we design a data structure whose space usage automatically improves with the compressibility of the graph at hand, while efficiently supporting navigational operations (simulating adjacency-list access). Specifically, we show that the space usage approaches the *instance-optimal* space when the graph is drawn according to the classic Barabási-Albert model of preferential-attachment graphs. Our data-structure techniques also work for arbitrary graphs, guaranteeing a size asymptotically no larger than an entropy-compressed edge list. A key technical contribution is the careful analysis of the instance-optimal space usage.

1. Introduction

In this paper, we design a compressed representation for graphs generated according to the Barabási-Albert model of preferential attachment approaching the instance-optimal space usage of $\lg(1/p)$ bits, for p the probability of the stored graph, that supports efficient operations within the same space. (Here and throughout, $\lg = \log_2$.)

The motivation and techniques of our work span the fields of information theory, compression algorithms, and succinct data structures. Information theory studies random processes (*sources*) for generating combinatorial objects, as a way to quantify the intrinsic information content in an object x. Compression methods are algorithms to efficiently construct representations of x that come close to this lower bound for the size of representations. Succinct data structures aim to support efficient queries for an object $x \in \mathcal{X}$ using $\lg|\mathcal{X}|(1+o(1))$ bits of space. This space usage is asymptotically optimal in the *worst case* over $\mathcal{X}$, but can, in principle, be improved to $\lg(1/\mathbb{P}[x])$ bits of space when the object is drawn randomly from $\mathcal{X}$ with probability $\mathbb{P}[x]$. *Universal (source) codes*, as studied in information theory, approximate that space usage over a whole class of random sources "automatically", i.e., *without knowing* the actual probabilities $\mathbb{P}[x]$. In some restricted cases (namely strings [FM00] and trees [MNSBW21]), such universal codes have successfully been augmented with efficient query support. When applied to graphs, information theory, compression algorithms, and succinct data structures all are, by comparison,

*University of Liverpool, UK, {ziad.ismaili-alaoui, n.namrata, sebastian.wild}@liverpool.ac.uk
†University of Marburg, Germany, wild@informatik.uni-marburg.de

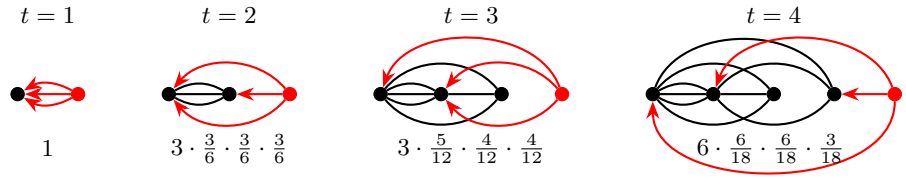

Figure 1: The Barabási-Albert $\mathrm{PA}(M; n)$ model of iteratively growing graphs with $M = 3$ for $n = 4$ time steps. The probability of the shown choices of M targets are given for each time step, yielding $\mathbb{P}[G_4] = 5/864$ overall. The information content is thus $\lg(1/\mathbb{P}[G_4]) \approx 7.43$ bits; the degree-entropy is $H_0^{\deg}(G_4) \approx 15.37$.

still in their infancy. A short survey of existing work on graphs from these angles is given in Section 1.2.

A classical notion of compressibility of a text T is its (zeroth-order) *empirical entropy* $H_0(t)$, (formally defined in Section 2). $H_0(T)$ gives a lower bound for the size of any representation that independently encodes individual characters of T, such as the well-known Huffman codes. It is also the entropy rate of the maximum-likelihood memoryless source for T, or the entropy of the character distribution obtained by Bayesian inference, i.e., starting with a uniform prior distribution, for each $T[i]$, update our prior (to a Beta distribution) to make $T[i]$ more likely to appear in the future.

Several competing notions of empirical entropies for trees have been proposed [HLSB20, HLSB19, KYS09] (see also [SB23, Part II]); for graphs, the grounding in information theory is weaker still. A simple and intuitive notion of empirical entropy for (directed) graphs G can be obtained as follows: Concatenate the adjacency lists (the out-neighbours) of all vertices and consider the empirical entropy $H_0(A)$ of the resulting *adjacency string* $A = A(G)$ over the alphabet $V(G)$. Observe that each vertex $v \in V(G)$ appears exactly $d_{\mathrm{in}}(v)$ times (for $d_{\mathrm{in}}(v)$ the in-degree of v) in $A(G)$; hence we call this quantity $H_0^{\deg}(G)$, the *degree entropy* of G. For ease of presentation, we focus on graphs where the out-degree of all vertices is the same and a canonical order of the vertices is given; then it is easy to reconstruct G from $A(G)$. Note that $H_0^{\deg}(G) = H_0(A(G)) \leq m \lg n$, where the latter is the number of bits used by an uncompressed adjacency list. (Here $n = |V(G)|$ is the number of vertices and $m = |E(G)|$ the number of edges.)

We will show that despite its simplicity, $H_0^{\deg}(G)$ can be a meaningful benchmark for graph compression. We point out here that this is at least partially surprising since $A(G)$ fixes a specific ordering (and hence, naming) of vertices and edges. We (almost always) consider the graph unchanged when (1) the edges are listed in different order, and, in many applications, (2) when arbitrary renaming of vertices occurs. In the latter case, we only seek to preserve the *structure* of a graph (i.e., the *unlabelled graph* or the equivalence class under relabelling), e.g., to encode labels separately. In addition, *undirected* edges are given a direction by listing them in $A(G)$. So, in general $A^{-1}(G)$ is far from unique and $H_0^{\deg}(G)$ is *not* in general a lower bound for the description length for G. This complication is indicative of what makes graph compression harder than text compression.

As a challenging testbed for compressed graph data structures, we hence consider *unlabelled*, *undirected* graphs from a highly non-uniform distribution. Specifically, we assume the *Barabási-Albert model* [BA99] of random *preferential-attachment graphs*. It generates a graph iteratively as follows (Figure 1). At each time step t, we add one new vertex v_t and draw M random neighbours for v_t, where M is a fixed parameter of the model. The M neighbours $a_{t,1}, \ldots, a_{t,M}$ are chosen, independently and with repetitions allowed, from the existing vertices $v_0, \ldots, v_{t-1}$ with probability *proportional* to the (current) *degree* of vertex v_i, $\mathbb{P}[u_{t,j} = v_i] \propto d_t(v_i)$. This

rule follows the *Matthew principle* (rich vertices get richer) and produces graphs where the degree-distribution follows a power law (asymptotically, the probability of a vertex to collect total degree k is proportional to k^{-3}). The heavy-tailed power-law degree distribution is widely observed in application domains with (approximately) scale-free graphs and makes compression interesting and non-trivial. The Barabási-Albert model is also a natural analogue of Bayesian inference of character distributions: whenever we see a vertex as the target of an edge, we consider this target as more likely for future choices.

1.1. Our Results

Suppose G is a graph generated by the Barabási-Albert model, and let $\mathbb{P}[G]$ be the probability for G to arise in this process (where we assume that M is a fixed parameter and we stop the generation process after the nth vertex). Any encoding for graphs that is simultaneously optimal for all such graphs must use (close to) $\lg(1/\mathbb{P}[G])$ bits of space to encode G. Our first set of results relate the amount of information in G, $\lg(1/\mathbb{P}[G])$, to the empirical degree-entropy of G, $H_0^{\deg}(G)$: despite their seemingly unrelated origins, the two quantities *coincide* up to an error term.

Theorem 1.1 (Instance-specific lower bound): *Let G be a labelled graph generated by preferential attachment, $G \sim \mathrm{PA}(M; n)$, and further assume that apart from a fixed-size seed graph, the graph G is simple. Then*

$$\lg(1/\mathbb{P}[G]) \;=\; H_0^{\deg}(G) \,\pm\, O(nM \lg M). \qquad \triangleleft$$

Note that $O(nM \lg M)$ is a lower order term for typical graphs when M is not too big, that is $\lg M = o(\lg n)$. The analysis (Section 3) uses Gibbs' inequality to compare different notions of empirical entropy, a trick that might be of independent interest.

For *unlabelled* graphs G, the bound reduces by $n \lg n$ bits:

Theorem 1.2 (Unlabelled lower bound): *Let G be as in Theorem 1.1 and consider its structure $S = S(G)$, i.e., a random* unlabelled *graph generated by preferential attachment. Then*

$$\lg(1/\mathbb{P}[S]) \;\geq\; H_0^{\deg}(S) - n \lg n \;-\; O(nM \lg M). \qquad \triangleleft$$

(Note that $H_0^{\deg}(G) = H_0^{\deg}(S(G))$.) It may seem obvious that the bound would reduce precisely by the $\lg(n!) \sim n \lg n$ bits needed to store n vertex labels since any unlabelled graph can correspond to at most $n!$ labelled graphs; however, we point out that (1) for general distributions over labelled graphs, it is not true that these $n!$ labelled graphs are equally likely, and (2) a strictly higher lower bound is the actual truth for certain S, e.g., graphs with linear diameter. We obtain a tight bound in Section 3, but its general growth with n is opaque. We do not know whether the bound in Theorem 1.2 is tight.

The degree entropy thus precisely captures the asymptotic information content of a preferential-attachment graph. We next show that we can represent G using close to $H_0^{\deg}(G)$ bits of space and support efficient queries on the compressed representation.

Theorem 1.3 (Ultrasuccinct preferential-attachment graphs): *Let G be obtained from $\mathrm{PA}(M; n)$. There are data structures that support operations finding (1) the ith in-neighbour, (2) the ith out-neighbour, (3) the degree of a vertex, and (4) testing adjacency, all in $O(\lg n)$ time, in the following total space:*

(a) *Storing the labelled graph G uses $H_0^{\deg}(G) + o(Mn)$ bits of space.*

(b) *Storing the unlabelled graph $S = S(G)$ uses at most $H_0^{\deg}(S)(1 - \frac{1}{M}) + 2n + o(Mn) \leq (M-1)n \lg n + 2n + o(Mn)$ bits of space.* ◁

By Theorem 1.1, the space for (a) is asymptotically optimal. The space for (b) almost matches Theorem 1.2; in particular, it does so on average: $\frac{1}{M} H_0^{\deg}(S)$ is close to $n \lg n$ unless S is very compressible. (Note that $A(S)$ has Mn characters, so $H_0^{\deg}(S) = Mn \cdot H_0^{\mathrm{pc}}(A(S))$ and $\frac{1}{M} H_0^{\deg}(S) = n \cdot H_0^{\mathrm{pc}}(A(S))$, for H_0^{pc} the *per-character entropy*; $H_0^{\mathrm{pc}}(A(S))$ is always between 0 and $\lg n$.)

The labelled graph data structure uses an entropy-compressed wavelet tree to represent the string $A(G)$. To reduce the space for an unlabelled graph, our data structure uses one outgoing edge per vertex (except v_0) as a *parent edge* in an ordinal tree. These can be stored using a succinct tree data structure using only 2 bits per edge if we use the tree indices to identify the graph vertices. The remaining out-neighbours form string A', which replaces A and is now n entries shorter, saving up to $\lg n$ bits each. By judiciously choosing which edges to turn into tree edges, we can moreover *retain* the *compressibility* of A'; we show that our choice implies that the per-character entropy in A' is never larger than that of A.

The unlabelled-graph data structure assigns new labels to the vertices of the graph, which have to be used in queries; the relabelling function can be provided during construction, but cannot be stored as part of the data structure in the stated space. A use case for the unlabelled-graph data structure is any task where we only need to identify a small subset of vertices; e.g., computing network-analysis metrics such as betweenness centrality for a subset of important vertices. We then store the names of important vertices in addition to the data structure for $S(G)$.

1.2. Related Work

The closest work to ours is the analysis of $\mathrm{PA}(M; n)$ by Łuczak et al. [LMS19a, LMS19b]. They compute the entropy of the distribution $G \sim \mathrm{PA}(M; n)$, $\mathbb{E}[\lg(1/\mathbb{P}[G])] = (M-1)n \lg n \cdot (1 + o(1))$ and (only in the conference version [LMS19b]) describe a compression algorithm achieving this entropy in expectation up to a redundancy of $O(n \lg \lg n)$. Our work improves their compression method to redundancy $O(n)$ (using a succinct tree instead of their backtracking numbers), provides instance-specific bounds, and supports efficient operations on the compressed representation. (It is not clear whether their compression algorithm can be turned into an efficient compressed data structure.)

Many more specialized models of random graphs have been suggested and studied. An example that also features a compressed data structure is the geometric inhomogeneous random graph (GIRG) model [BKL19]. The expected compressed space matches the entropy up to constant factors. It is not clear if instance-optimal space is easily achievable. As stated, the compression method requires a geometric realization of the graph as input.

Another related area is applied graph compression, where methods are evaluated empirically on benchmarks from specific domains. As a representative example, we mention the webgraph framework [BV04], which provides space-efficient data structures tuned to large web graphs.

The idea to use wavelet trees for the edges of a graph is discussed in Navarro's survey on wavelet trees [Nav14] (and may be folklore). This approach alone is inherently labelled and thus cannot escape the lower bound of Theorem 1.1. The closest prior work to our data structure is the GLOUDS graph representation of Fischer and Peters [FP16]. GLOUDS partitions the edges into tree edges and non-tree edges using a standard breadth-first search to obtain an ordinal tree; it uses a custom representation for the tree, as well. GLOUDS does not achieve optimal space for $G \sim \mathrm{PA}(M; n)$ with constant M. In its original form it also does not adapt

to $H_0^{\deg}(G)$ (although entropy compressing their array H would move towards the latter).

There are many further works that are related in spirit to our work, but with a distinctly different angle. For certain specific classes of graphs, *succinct* data structures have been designed: interval graphs [ACJRS20, HMN$^+$20], chordal graphs [MW18], permutation graphs [TWZ23], circle graphs [ACJ$^+$22], bounded clique-width graphs [CJSRS24], series-parallel graphs [CJSRS23]. For representing any graph from a class of graphs with g_n graphs on vertex set $[n]$, these data structures use $\lg(g_n)(1+o(1))$ bits of space for representing one such graph; this invariably corresponds to a uniform distribution over the class of graphs under consideration. Supported operations vary, but always include finding neighbours of a given vertex.

For trees, data structures beating the worst-case optimal space on compressible inputs have been considered. Ultrasuccinct trees [JSS12] adapt to the degree entropy of a tree. Hypersuccinct trees [MNSBW21] have been shown to simultaneously yield optimal space for a large variety of entropy measures, both empirical (such as degree entropy) and entropy rates of random tree sources. Here we lift these results to "ultrasuccinct graphs".

On the information theory of structures (unlabelled graphs), a line of work studied the entropy of random unlabelled graphs. Apart from the works of Łuczak et al. [LMS19a, LMS19b] on preferential attachment graphs, small-world graphs [KLPS22], the Erdős-Renyi graphs [CS12], and a vertex-copying model [TMS20] have been studied.

Lossless compression of graphs is surveyed by Besta and Hoefler [BH19]; see also [BLKB21]. Typical examples consider an application and exploit specifics of graphs arising there. A statistical model of data and efficient queries are often not available.

2. Preliminaries

In this section, we collect some definitions and known results that we build on in this work. We write $[a..b]$ for $\{a, a+1, \ldots, b\}$ and abbreviate $[n] = [1..n]$; also $[a..b) = [a..b-1]$. We use standard graph terminology; specifically, for a graph G and a vertex $v \in V(G)$, we write $d(v) = d_G(v)$ for the degree of v in G, i.e., the number of (potentially parallel) edges incident at v. For directed graphs (digraphs), we distinguish in-degree and out-degree: $d(v) = d_{\text{in}}(v) + d_{\text{out}}(v)$; we also write $N_{\text{out}}(v)$ for the out-neighbourhood of v, i.e., the (multi)set of vertices w for which there is a (bundle of) edge(s) vw in $E(G)$. We call a DAG *M-out-regular* if $d_{\text{out}}(v) = M$ for all vertices $v \in V \setminus \{v_0\}$.

2.1. Empirical Entropy

For a text $T[1..n]$ over alphabet $\Sigma = [1..\sigma]$, the *zeroth-order empirical entropy* $H_0(T)$ is given by

$$H_0(T) \;=\; \sum_{c=1}^{\sigma} |T|_c \lg\Big(\frac{n}{|T|_c}\Big),$$

where $|T|_c = |\{i \in [1..n] : T[i] = c\}|$ denotes the number of occurrences of c in T. Additionally, we define $H_0^{\text{pc}}(T) = H_0(T)/|T|$ to be the *per-character zeroth-order empirical entropy* of T. As is standard, we set $0 \lg(n/0) := 0$ for notational convenience.

2.2. Preferential-Attachment Graphs

We consider the classical Barabási-Albert model of generating a random (undirected) graph (with parallel edges); see Figure 1. Given a target size n and a parameter $M \in \mathbb{N}$, we grow

a graph G iteratively, where vertex v_t arrives at time t, for $t = 0, \ldots, n$. We denote by G_t the graph after this arrival at time t; the vertex set is $V(G_t) = \{v_0, \ldots, v_t\}$. G_0 is the single isolated vertex v_0; G_1 has M parallel edges between v_0 and v_1. G_t for $t \geq 2$ results from G_{t-1} by adding v_t and M edges from v_t to targets $a_{t,1}, \ldots, a_{t,M} \in V(G_{t-1}) = \{v_0, \ldots, v_{t-1}\}$, where the $a_{t,1}, \ldots, a_{t,M}$ are mutually independent and identically distributed with

$$\mathbb{P}[a_{t,1} = v_\ell] \;=\; \cdots \;=\; \mathbb{P}[a_{t,M} = v_\ell] \;=\; \frac{d_{G_{t-1}}(v_\ell)}{2|E(G_{t-1})|} \qquad (\ell = 0, \ldots, t-1).$$

Note that if we orient the edges from new to old vertices in $G_n \sim \mathrm{PA}(M; n)$ (in the order they arrived to the graph, i.e., from large to small t), we obtain a directed acyclic graph with uniform out-degrees: $d_{\mathrm{out}}(v_t) = M$ for $t \in [1..n]$ ($d_{\mathrm{out}}(v_0) = 0$).

We will abbreviate $d_t(v) = d_{G_t}(v)$. Note that $|E(G_t)| = tM$ and $|V(G_t)| = t + 1$. We write $G_n \sim \mathrm{PA}(M; n)$ to indicate that G_n has been randomly chosen according to this preferential-attachment process.

For $G \sim \mathrm{PA}(M; n)$, we obtain $H_0^{\deg}(G)$ as $H_0^{\deg}(G) = H_0(A(G))$ where

$$A(G) = [a_{1,1}, \ldots, a_{1,M}, \ \ldots, \ a_{n,1}, \ldots, a_{n,M}].$$

Let $S(G)$ denote the *structure* of G, i.e., the class of graphs which are isomorphic to G. We also refer to these as *unlabelled graphs*. We define the unlabelled graph distribution $PA^u(M; n)$ as the probability distribution on the family of $S(G)$, where the probability of each class is the sum of the probabilities that a labelled version of $S(G)$ is $\mathrm{PA}(M; n)$, that is,

$$\sum_{H \in S(G)} \mathbb{P}[H = \mathrm{PA}(M; n)].$$

2.3. Succinct Data Structures

For the reader's convenience, we collect used results on succinct data structures here. First, we cite the compressed bit vectors of Pătraşcu [Păt08].

Lemma 2.1 (Compressed bit vector): *Let $B[1..n]$ be a bit vector of length n, containing m 1-bits. For any constant $c > 0$, there is a data structure using $\lg \binom{n}{m} + O(\frac{n}{\lg^c n}) \leq m \lg(\frac{n}{m}) + O(\frac{n}{\lg^c n} + m)$ bits of space that supports in $O(1)$ time operations (for $i \in [1..n]$):*

- $\mathsf{access}(B, i)$*: return $B[i]$, the bit at index i in B;*
- $\mathsf{rank}_\alpha(B, i)$*: return the number of bits with value $\alpha \in \{0, 1\}$ in $B[1..i]$;*
- $\mathsf{select}_\alpha(B, i)$*: return the index of the ith bit with value $\alpha \in \{0, 1\}$.* ◁

Using wavelet trees, we can support rank and select queries on arbitrary static strings/sequences while compressing them to zeroth-order empirical entropy.

Lemma 2.2 (Wavelet tree [Nav14]): *Let $S[1..n]$ be a array with entries $S[i] \in \Sigma = [1..\sigma]$. There is a data structure using $H_0(S) + o(n)$ bits of space that supports the following queries in $O(\lg \sigma)$ time (without access to S at query time):*

- $\mathsf{access}(S, i)$*: return $S[i]$, the symbol at index i in S;*
- $\mathsf{rank}_\alpha(S, i)$*: return the number of indices with value $\alpha \in \Sigma$ in $S[1..i]$;*
- $\mathsf{select}_\alpha(S, i)$*: return the index of the ith occurrence of value $\alpha \in \Sigma$ in S.* ◁

Proof: Many variants of wavelet trees are possible [Nav14]; for our result, we use the plain encoding of the characters with $\lceil \lg \sigma \rceil$ bits each (balanced wavelet tree). All levels except the last are full, so we can concatenate all bitvectors of nodes in the wavelet tree in level order (pointerless wavelet tree [Nav14, §2.3]) and support navigation up and down the tree. Moreover, using Lemma 2.1 for this bitvector of length at most $\lceil \lg \sigma \rceil n$ yields space $H_0(S) + o(n)$ ([Nav14, §3.1]). □

We lastly need a succinct data structure for ordinal trees. Several options exist for the basic queries we need [BDM+05, GRR06]; (see [HMN+20, App. A] for a comprehensive review).

Lemma 2.3 (Succinct ordinal trees): *Let T be an ordinal tree on n vertices. There is a data structure using $2n + o(n)$ bits of space that supports the following queries in $O(1)$ time (where nodes are identified with their preorder index in T):*

- $\mathsf{parent}(T, v)$*: return the parent of v in T;*
- $\mathsf{degree}(T, v)$*: return the number of children of v in T;*
- $\mathsf{child}(T, v, i)$*: return the ith child of v in T.* ◁

3. Space Lower Bound

In this section, we derive the instance-specific lower bounds for preferential-attachment graphs. Section 3.1 gives the main result for *labelled* graphs. Appendix A gives the proof for the *unlabelled* case.

Let $G \sim \mathrm{PA}(M; n)$ be a labelled preferential-attachment graph. Recall that G is drawn iteratively, where vertex v_t arrives at time t, for $t = 1, \ldots, n$. The graph after the arrival of v_t is G_t, where $V(G_t) = \{v_0, \ldots, v_t\}$. The total number of edges in $G = G_n$ is $m = Mn$.

For time $t = 1, \ldots, n$, let $N_{\mathrm{out}}(v_t)$ denote the random multiset of size M that contains the M out-neighbours of v_t randomly sampled from the set $V(G_{t-1}) = \{v_0, v_1, \ldots, v_{t-1}\}$. The vertex v_t attaches to a vertex in $V(G_{t-1})$ with probability proportional to its degree at time $t - 1$:

$$P_i^{(t)} \;=\; \mathbb{P}[v_i \in N_{\mathrm{out}}(v_t) \,|\, G_{t-1}] \;=\; \frac{d_{t-1}(v_i)}{2(t-1)M}\,. \tag{1}$$

Multinomial distribution of $N_{\mathsf{out}}(v_t)$. The frequencies of appearance of vertices as out-neighbours of v_t follow a multinomial distribution. The number of trials in this case is M, where each trial when v_t chooses the neighbour from the set $V(G_{t-1})$ is independent of each other. Also, each trial has t possible outcomes $v_0, v_1, \ldots v_{t-1}$, with probabilities as given by Equation (1). Therefore, we have (conditional on G_{t-1}), that $(C_0^{(t)}, \ldots, C_{t-1}^{(t)}) \sim \mathrm{Mult}(M; P_0^{(t)}, P_1^{(t)}, \ldots, P_{t-1}^{(t)})$, where $C_i^{(t)}$ denote the number of times the vertex v_i is chosen randomly as a neighbour of v_t at time t. Therefore,

$$\mathbb{P}[(C_0^{(t)}, \ldots, C_{t-1}^{(t)}) = (c_0, \ldots c_{t-1}) \,|\, G_{t-1}] \;=\; \binom{M}{c_0, \ldots, c_{t-1}} (P_0^{(t)})^{c_0} \ldots (P_{t-1}^{(t)})^{c_{t-1}}.$$

3.1. Proof of Theorem 1.1

In this subsection, we derive a lower bound on $\lg(1/\mathbb{P}[G])$ for $\mathbb{P}[G]$ the probability that a given graph $G \sim \mathrm{PA}(M;n)$ results from a preferential-attachment process $\mathrm{PA}(M;n)$.

Recall that $G = G_n$. Using the product rule, we get

$$\mathbb{P}[G] \;=\; \mathbb{P}[N_{\mathrm{out}}(v_n) \mid G_{n-1}] \cdot \mathbb{P}[N_{\mathrm{out}}(v_{n-1}) \mid G_{n-2}] \;\cdots\; \mathbb{P}[N_{\mathrm{out}}(v_2) \mid G_1] \cdot \mathbb{P}[G_1].$$

This implies

$$\begin{aligned}\lg\Big(\frac{1}{\mathbb{P}[G]}\Big) &= \sum_{t=2}^{n} \lg\Big(\frac{1}{\mathbb{P}[N_{\mathrm{out}}(v_t) \mid G_{t-1}]}\Big) \\ &= \sum_{t=2}^{n}\Bigg[\sum_{i=0}^{t-1} C_i^{(t)} \lg\Big(\frac{2(t-1)M}{d_{t-1}(v_i)}\Big) \;-\; \lg\Bigg(\binom{M}{C_0^{(t)},\ldots,C_{t-1}^{(t)}}\Bigg)\Bigg]. \end{aligned} \tag{2}$$

Consider $-\sum_{t=2}^{n} \lg\Big(\binom{M}{C_0^{(t)},\ldots,C_{t-1}^{(t)}}\Big)$. We have

$$-\lg\Bigg(\binom{M}{C_0^{(t)},\ldots,C_{t-1}^{(t)}}\Bigg) \;=\; \lg\Bigg(\frac{C_0^{(t)}!\ldots C_{t-1}^{(t)}!}{M!}\Bigg) \;\geq\; \lg\Big(\frac{1}{M!}\Big) \;=\; -\lg(M!).$$

Now let us consider the first summand in (2),

$$\begin{aligned}&\sum_{t=2}^{n}\sum_{i=0}^{t-1} C_i^{(t)} \lg\Big(\frac{2(t-1)M}{d_{t-1}(v_i)}\Big) \\ &\quad= \sum_{t=1}^{n-1}\Bigg(\sum_{i=0}^{t} C_i^{(t+1)} \lg(2tM)\Bigg) \;-\; \sum_{t=1}^{n-1}\Bigg(\sum_{i=0}^{t} C_i^{(t+1)} \lg(d_t(v_i))\Bigg). \end{aligned} \tag{3}$$

Let us analyse the first summand. With $\sum_{i=0}^{t} C_i^{(t+1)} = M$, we find

$$\begin{aligned}\sum_{t=1}^{n-1}\Bigg(\sum_{i=0}^{t} C_i^{(t+1)} \lg(2tM)\Bigg) &= \sum_{t=1}^{n-1} \lg(2tM)\Bigg(\sum_{i=0}^{t} C_i^{(t+1)}\Bigg) \\ &= M(n-1)\lg(2M) + M\lg((n-1)!).\end{aligned}$$

Now consider the second summand in (3). Swapping the order of summation yields

$$-\sum_{t=1}^{n-1}\Bigg(\sum_{i=0}^{t} C_i^{(t+1)} \lg(d_t(v_i))\Bigg) \;=\; -\sum_{i=0}^{n-1}\Bigg(\sum_{t=i+1}^{n-1} C_i^{(t+1)} \lg(d_t(v_i))\Bigg).$$

Let $T_i = \{t \geq 2 : C_i^{(t)} > 0\}$ denote the set of timestamps such that the vertex v_i is selected as an out-neighbour; we have $T_i = \{t_1^{(i)},\ldots,t_{d_{\mathrm{in}}(v_i)}^{(i)}\}$ with $t_1^{(i)} \leq \cdots \leq t_{d_{\mathrm{in}}(v_i)}^{(i)}$, i.e., just before time $t_k^{(i)}$, v_i's total degree was $d_{t_k^{(i)}-1}(v_i) = M + k - 1$.[1] So far, the analysis works for general

[1] For ease of notation, we do not count the M edges from v_1 to v_0 in $d_{\mathrm{in}}(v_0)$ here; then for all vertices, we have $d(v_i) = d_{\mathrm{in}}(v_i) + M$. That is also why we only include times $t \geq 2$ in T_i.

graphs; for the following simplification, we assume that no parallel edges[2] are added in the graph (for $t \geq 2$). Then $C_i^{(t)} \leq 1$ and the summand becomes

$$\begin{aligned}
-\sum_{i=0}^{n-1} \sum_{\substack{t=i+1 \\ t\geq 2}}^{n} C_i^{(t)} \lg(d_{t-1}(v_i)) &= -\sum_{i=0}^{n-1} \sum_{k=1}^{d_{\text{in}}(v_i)} \lg(M+k-1) \\
&= -\sum_{i=0}^{n-1} \left(\sum_{k=1}^{d_{\text{in}}(v_i)+M-1} \lg(k) - \sum_{k=1}^{M-1} \lg(k) \right) \\
&= -\sum_{i=0}^{n-1} \lg((d(v_i)-1)!) \;+\; n\lg((M-1)!).
\end{aligned}$$

Therefore, Equation (2) becomes

$$\begin{aligned}
\lg\left(\frac{1}{\mathbb{P}[G]}\right) &\geq M(n-1)\lg(2M) + M(\lg((n-1)!)) \\
&\quad - \sum_{i=0}^{n-1} \lg((d(v_i)-1)!) \;+\; n\lg((M-1)!) \\
&\quad - (n-1)\lg(M!) \\
&= Mn\lg(2Mn) \;-\; \sum_{i=0}^{n-1} d(v_i)\lg(d(v_i)) \;\pm\; O(Mn)
\end{aligned}$$

(using $d(v) = d_{\text{in}}(v) + M$)

$$= Mn \underbrace{\sum_{i=0}^{n-1} \frac{d_{\text{in}}(v_i)}{Mn} \lg\left(\frac{2Mn}{d(v_i)}\right)}_{(*)} \;-\; \underbrace{\sum_{i=0}^{n-1} M\lg(d(v_i))}_{(\dagger)} \;\pm O(Mn)$$

(bounding $(*)$ by Gibbs' inequality, and $(\dagger)$ by the log-sum inequality)

$$\begin{aligned}
&\geq Mn \sum_{i=0}^{n-1} \frac{d_{\text{in}}(v_i)}{Mn} \lg\left(\frac{Mn}{d_{\text{in}}(v_i)}\right) \;\pm\; O(Mn\lg M) \\
&= H_0^{\deg}(G) \;\pm\; O(Mn\lg M). \qquad (4)
\end{aligned}$$

This proves Theorem 1.1.

(The proof of Theorem 1.2 is given in Appendix A.)

4. Data Structures

In this section, we describe data structures that can represent a directed graph generated by the preferential-attachment process in compressed form, while allowing for efficient navigational queries without decompression.

For labelled graphs, we only use the wavelet tree portion of the data structure described in the following. We omit the simple modifications here and present the data structure for unlabelled graphs in detail; (see also [Nav14, §5.3] for the labelled case).

[2]We note that G_M necessarily contains parallel edges, and our assumptions concern the edges chosen for $t > M$. The contribution of these parallel edges in the calculation below can be shown to be $O(M)$ overall, so we ignore their presence for legibility.

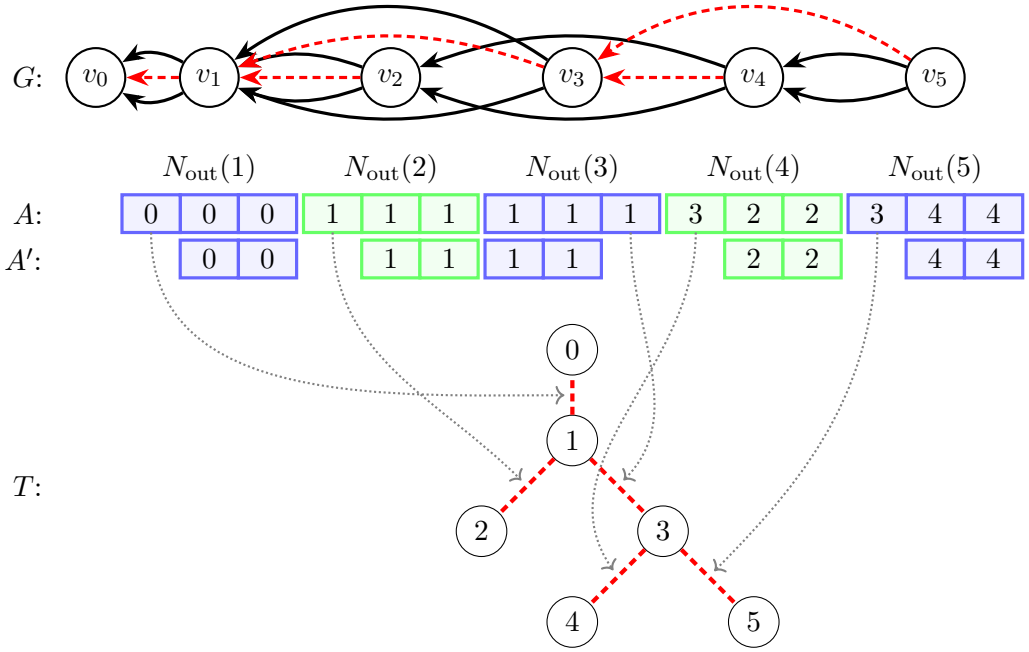

Figure 2: Illustration of the construction of T, A and A' on a sample preferential-attachment graph where $n = 5$ and $M = 3$. The dashed red edges in G represent the edges that are represented in T. Note that the vertices are labelled by their preorder index in T.

4.1. Construction

Let $G' \sim \mathrm{PA}(M; n)$ be generated by the Barabási-Albert model. By iteratively "peeling off" degree-M vertices, we uniquely obtain the directed version $G = DAG(G')$ [LMS19a].

We now sort vertices by indegree and construct a tree T informally by choosing, for each vertex v_t in G (except for v_0), a parent among its out-neighbours with minimal (in-)degree. Formally, define a bijective *rank* function $\sigma : \{v_0, \ldots, v_n\} \to [0..n]$ such that $d_{\text{in}}(u) > d_{\text{in}}(v)$ implies $\sigma(u) > \sigma(v)$. Construct a tree T with $V(T) = V(G)$ and

$$E(T) \;=\; \bigcup_{t=1}^{n} \Big\{ \big(v_t,\ \operatorname*{arg\,min}_{w \in N_{\text{out}}(v_t)} \sigma(w)\big) \Big\}.$$

Observe that T is indeed a tree since it consists of $n+1$ vertices and n edges and is acyclic by definition. Vertex v_0 is the root of T.

We will store T using Lemma 2.3, where operations refer to nodes via their preorder index. We hence perform a preorder traversal of T from its root, and relabel $v_t \in V(G)$ by its index in the traversal, starting from 0.

For each vertex $v \neq 0$ in G, define an order on its outgoing edges such that the edge (v, u), for some u, is the first if and only if $(v, u) \in E(T)$, and all other vertices (v, w), $u \neq w$, are ordered arbitrarily. Finally, construct a static array A' such that the $(i+1)$th ($i \in [1..M)$) out-neighbour of vertex j in G is stored in $A'[(j-1)(M-1)+i]$. We store A' in a wavelet tree using Lemma 2.2.

4.2. Operations

Building upon the aforementioned data structures, we present algorithms to efficiently handle a variety of navigational queries on G. Define $\mathsf{source}(i) = \lceil \frac{i}{M-1} \rceil$ as the source of the edge whose

target is stored in $A'[i]$.

Computing the ith out-neighbour of v. The ith *out-neighbour* of v in G can be determined as follows: if $i = 1$, it is the parent of v in T; otherwise, it is given by $A'[(v-1)(M-1)+i-1]$ (the $(i-1)$th element of $A'[(v-1)(M-1)+1..v(M-1)]$, which represents all but the first out-neighbour of v).

Computing the ith in-neighbour of v. To determine the ith *in-neighbour* of v, we distinguish two cases:

1. If v has at least i children in T, then its ith in-neighbour in G is simply its ith child in T.
2. Otherwise, if $i > \mathsf{degree}(T, v)$, we determine the $(i - \mathsf{degree}(T, v))$th occurrence of v in A'. The vertex whose neighbourhood includes this occurrence is found by computing $\mathsf{source}(\mathsf{select}_v(S, i - \mathsf{degree}(T, v)))$.

Using these two operations, we can also iterate through the neighbourhoods or return all (in- resp. out-) neighbours.

Computing the in-degree of v. The in-degree of a vertex v can be determined by summing the number of its children in T (computable by $\mathsf{degree}(T, v)$) and its occurrences in A' (computable by $\mathsf{rank}_v(A', n(M-1))$.

Determining if u and v are adjacent. If u and v are adjacent, either (1) u is the parent of v in T, $\mathsf{parent}(T, v) = u$, (2) vice versa, $\mathsf{parent}(T, u) = v$, or (3) v occurs in $A'[(u-1)(M-1)+1..u(M-1)]$ or (4) vice versa, u occurs in $A'[(v-1)(M-1)+1..v(M-1)]$. (If either of them is v_0, two cases need not be checked.) Conditions (1) and (2) are directly supported on T; for (3) and (4), we can use rank: If $\mathsf{rank}_v(A', u(M-1)) - \mathsf{rank}_v(A', (u-1)(M-1)) \geq 1$, then v occurs at least once, so v and u are adjacent. (4) is similar.

To conclude the proof of Theorem 1.3, it remains to analyse the space usage of our data structure; details are given in Appendix B.

Remark 4.1 (General graphs): We point out that the data-structure techniques above can be extended to general graphs. Using a compressed bit vector to mark where the next neighbourhoods begin, we can support the same queries with $n \lg(m/n) + O(n)$ extra space. ◁

5. Conclusion

We designed a compressed representation for graphs generated by the Barabási-Albert model of random preferential-attachment graphs approaching the instance-optimal $\lg(1/\mathbb{P}[G])$ bits of space, where $\mathbb{P}[G]$ is the probability for the graph to arise in the model. We further related $\lg(1/\mathbb{P}[G])$ to the empirical degree-entropy of G, $H_0^{\deg}(G)$; they coincide up to an error term in both labelled and unlabelled graphs. Our compressed representation supports navigational queries in $O(\log n)$ time and can simulate access to an adjacency-list representation.

Future work might study other models of preferential attachment. For example, the model introduced by Cooper and Frieze [CF03], where the number of edges added follows a given distribution. One can also consider a non-linear model in which the probability of a vertex, when M edges are added, being chosen is proportional to its degree raised to some power α. (Here $\alpha = 1$.)

More broadly speaking, this work merely made initial strides into data structures that adapt to information-theoretic measures of compressibility in graphs, leaving many avenues for future work open. For example, unlike in trees, a convincing notion of higher-order entropy for graphs is not yet emerging, let alone data structures approaching them.

References

[ACJ+22] Hüseyin Acan, Sankardeep Chakraborty, Seungbum Jo, Kei Nakashima, Kunihiko Sadakane, and Srinivasa Rao Satti. Succinct navigational oracles for families of intersection graphs on a circle. *Theoretical Computer Science*, 928:151–166, September 2022. doi:10.1016/j.tcs.2022.06.022.

[ACJRS20] Hüseyin Acan, Sankardeep Chakraborty, Seungbum Jo, and Srinivasa Rao Satti. Succinct encodings for families of interval graphs. *Algorithmica*, 83(3):776–794, April 2020. doi:10.1007/s00453-020-00710-w.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999. doi:10.1126/science.286.5439.509.

[BDM+05] David Benoit, Erik D. Demaine, J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005. doi:10.1007/s00453-004-1146-6.

[BH19] Maciej Besta and Torsten Hoefler. Survey and taxonomy of lossless graph compression and space-efficient graph representations, 2019. arXiv:1806.01799.

[BKL19] Karl Bringmann, Ralph Keusch, and Johannes Lengler. Geometric inhomogeneous random graphs. *Theoretical Computer Science*, 760:35–54, February 2019. doi:10.1016/j.tcs.2018.08.014.

[BLKB21] Giorgos Bouritsas, Andreas Loukas, Nikolaos Karalias, and Michael Bronstein. Partition and code: learning how to compress graphs. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 18603–18619. Curran Associates, Inc., 2021.

[BV04] P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, WWW04, page 595–602. ACM, May 2004. doi:10.1145/988672.988752.

[CF03] Colin Cooper and Alan Frieze. A general model of web graphs. *Random Structures & Algorithms*, 22(3):311–335, 2003.

[CJSRS23] Sankardeep Chakraborty, Seungbum Jo, Kunihiko Sadakane, and Srinivasa Rao Satti. Succinct data structures for sp, block-cactus and 3-leaf power graphs. *International Journal of Foundations of Computer Science*, page 1–18, April 2023. doi:10.1142/s012905412341006x.

[CJSRS24] Sankardeep Chakraborty, Seungbum Jo, Kunihiko Sadakane, and Srinivasa Rao Satti. Succinct data structures for bounded clique-width graphs. *Discrete Applied Mathematics*, 352:55–68, July 2024. doi:10.1016/j.dam.2024.03.016.

[CS12] Yongwook Choi and Wojciech Szpankowski. Compression of graphical structures: Fundamental limits, algorithms, and experiments. *IEEE Transactions on Information Theory*, 58(2):620–638, February 2012. doi:10.1109/tit.2011.2173710.

[FM00] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Comput. Soc, 2000. doi:10.1109/sfcs.2000.892127.

[FP16] Johannes Fischer and Daniel Peters. Glouds: Representing tree-like graphs. *Journal of Discrete Algorithms*, 36:39–49, January 2016. doi:10.1016/j.jda.2015.10.004.

[GRR06] Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2(4):510–534, October 2006. doi:10.1145/1198513.1198516.

[HLSB19] Danny Hucke, Markus Lohrey, and Louisa Seelbach Benkner. Entropy bounds for grammar-based tree compressors. In *IEEE International Symposium on Information Theory (ISIT)*, page 1687–1691. IEEE, July 2019. doi:10.1109/isit.2019.8849372.

[HLSB20] Danny Hucke, Markus Lohrey, and Louisa Seelbach Benkner. A comparison of empirical tree entropies. In *String Processing and Information Retrieval (SPIRE)*, page 232–246. Springer, 2020. doi:10.1007/978-3-030-59212-7_17.

[HMN+20] Meng He, J. Ian Munro, Yakov Nekrich, Sebastian Wild, and Kaiyu Wu. Distance oracles for interval graphs via breadth-first rank/select in succinct trees. In *International Symposium on Algorithms and Computation (ISAAC)*, LIPIcs, pages 25:1–25:18. Schloss Dagstuhl, 2020. arXiv:2005.07644.

[JSS12] Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees with applications. *Journal of Computer and System Sciences*, 78(2):619–631, March 2012. doi:10.1016/j.jcss.2011.09.002.

[KLPS22] Ioannis Kontoyiannis, Yi Heng Lim, Katia Papakonstantinopoulou, and Wojtek Szpankowski. Compression and symmetry of small-world graphs and structures. *Communications in Information and Systems*, 22(2):275–302, 2022. doi:10.4310/cis.2022.v22.n2.a5.

[KYS09] John C. Kieffer, En-Hui Yang, and Wojciech Szpankowski. Structural complexity of random binary trees. In *2009 IEEE International Symposium on Information Theory*. IEEE, jun 2009. doi:10.1109/isit.2009.5205704.

[LMS19a] Tomasz Luczak, Abram Magner, and Wojciech Szpankowski. Asymmetry and structural information in preferential attachment graphs. *Random Structures & Algorithms*, 55(3):696–718, March 2019. doi:10.1002/rsa.20842.

[LMS19b] Tomasz Luczak, Abram Magner, and Wojciech Szpankowski. Compression of preferential attachment graphs. In *IEEE International Symposium on Information Theory (ISIT)*, page 1697–1701. IEEE, July 2019. doi:10.1109/isit.2019.8849739.

[MGSS17] Abram Magner, Ananth Grama, Jithin Sreedharan, and Wojciech Szpankowski. Recovery of vertex orderings in dynamic graphs. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 1563–1567. IEEE, 2017.

[MNSBW21] J. Ian Munro, Patrick K. Nicholson, Louisa Seelbach Benkner, and Sebastian Wild. Hypersuccinct trees – new universal tree source codes for optimal compressed tree data structures and range minima. In P. Mutzel, R. Pagh, and G Herman, editors, *European Symposium on Algorithms (ESA)*, pages 70:1–70:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. *arXiv:2104.13457*, *doi:10.4230/LIPICS.ESA.2021.70*.

[MOA79] Albert W Marshall, Ingram Olkin, and Barry C Arnold. Inequalities: theory of majorization and its applications, 1979.

[MW18] J. Ian Munro and Kaiyu Wu. Succinct data structures for chordal graphs. In *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, pages 67:1–67:12, 2018. *doi:10.4230/LIPIcs.ISAAC.2018.67*.

[Nav14] Gonzalo Navarro. Wavelet trees for all. *Journal of Discrete Algorithms*, 25:2–20, 2014.

[Păt08] Mihai Pătraşcu. Succincter. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, October 2008. *doi:10.1109/focs.2008.83*.

[SB23] Louisa Seelbach Benkner. *Combinatorial and information-theoretic aspects of tree compression*. PhD thesis, University of Siegen, 2023. *doi:10.25819/UBSI/10429*.

[TMS20] Krzysztof Turowski, Abram Magner, and Wojciech Szpankowski. Compression of dynamic graphs generated by a duplication model. *Algorithmica*, 82(9):2687–2707, April 2020. *doi:10.1007/s00453-020-00699-2*.

[TWZ23] Konstantinos Tsakalidis, Sebastian Wild, and Viktor Zamaraev. Succinct permutation graphs. *Algorithmica*, 85(2):509–543, 2023. *arXiv:2010.04108*, *doi:10.1007/s00453-022-01039-2*.

Appendix

A. Proof of Theorem 1.2 (Lower Bound for Unlabelled PA Graphs)

Assuming an undirected graph G has been generated by the Barabási-Albert process, we can uniquely reconstruct its directed version by iteratively "peeling" off degree-M vertices [LMS19a]. Note that the same is *not* true about the arrival times; in general the resulting DAG contains only partial information about the vertex arrival order: any linear extension of the partial order induced by the DAG is an admissible arrival order. Our analysis builds on results by Łuczak et al. who proved that (1) most PA graphs have $n! \cdot 2^{-O(n \lg \lg n)}$ admissible arrival orders [LMS19a, p. 715] and (2) that each of these arises from $\mathrm{PA}(M; n)$ with equal probability [LMS19a, Lem. 7]. We reproduce the relevant arguments here.

We define the admissible set $\mathrm{Adm}(S)$ of a given unlabelled graph S to be the set of all labelled graphs G with $S(G) = S$ such that G could have been generated according to the preferential-attachment model. We can also define $\mathrm{Adm}(G) := \mathrm{Adm}(S(G))$. For a graph G, we define $\Gamma(G)$ to be the set of permutations π such that $\pi(G) \in \mathrm{Adm}(G)$. Let $\mathrm{Aut}(G)$ be the automorphism group of G. For any G, we have the following equation [MGSS17]

$$|\mathrm{Adm}(G)| := \frac{|\Gamma(G)|}{|\mathrm{Aut}(G)|}. \tag{5}$$

For a graph $G \sim \mathrm{PA}(M; n)$ and $S(DAG(G))$, every possible way to order the vertices will result in the *same* probability for generating the resulting labelled $DAG(G)$ since we can reorder the numerators and denominators from (1).

Lemma A.1 ([LMS19a, Lem. 7]): *Let $H \sim \mathrm{PA}(M; n)$ for some $M \geq 1$. For any two graphs G, G' without parallel edges apart from the seed graph G_1 satisfying $S(DAG(G)) = S(DAG(G'))$, we have $\mathbb{P}(H = G) = \mathbb{P}(H = G')$.* ◁

For an unlabelled graph $S(G)$ without parallel edges, Lemma A.1 implies

$$\mathbb{P}[S(G)] \;=\; \mathbb{P}[G] \cdot |\mathrm{Adm}(S(G))|.$$

Since $|\mathrm{Adm}(G)| \leq n!$, the instance-specific lower bound for unlabelled $S(G)$ is

$$\lg\!\left(\frac{1}{\mathbb{P}[S(G)]}\right) \;=\; \lg\!\left(\frac{1}{\mathbb{P}[G]}\right) - \lg|\mathrm{Adm}(S(G))| \;\geq\; \lg(1/\mathbb{P}[G]) - \lg(n!),$$

which proves Theorem 1.2.

B. Space and Time Analysis of our Data Structure

In this section, we prove the space and time complexity of our preferential-attachment graph representation. Note that for labelled graphs, the result follows directly from the guarantees of wavelet trees: We use Lemma 2.2 to represent $A = A(G)$; since we have (by definition) that $H_0(A(G)) = H_0^{\deg}(G)$, Theorem 1.3–(a) follows.

For Theorem 1.3–(b), we first state the achieved results in terms of the empirical entropy of A':

Lemma B.1: *There exist data structures to represent T and A' in $H_0(A') + 2n + o(Mn)$ bits of space, while allowing for $\mathsf{N_{out}}(v, i)$, $\mathsf{N_{in}}(v, i)$, $\mathsf{degree}(v)$, and $\mathsf{adjacency}(u, v)$ to run in $\mathrm{O}(\lg n)$ time, and $\mathsf{N_{out}}(v)$ and $\mathsf{N_{in}}(v)$, in $\mathrm{O}(\lg n)$ time per neighbour.* ◁

Proof: As stated in Lemma 2.3, the tree T can be succinctly represented using $2n + o(n)$ bits, while supporting relevant tree operations in constant time. Since T encodes the first out-neighbour of each vertex in G, the remaining $n(M-1)$ edges in A' must be stored separately.

We store A' using a wavelet tree. By Lemma 2.2, this uses $H_0(A') + o(Mn)$ bits (since $|A'| = n(M-1)$), while allowing access, rank, and select queries in $\mathrm{O}(\lg \sigma)$ time, where σ, the alphabet size, is n here.

Overall, the data structures representing the preferential-attachment graph G occupy a total of $H_0(A') + 2n + o(Mn)$ bits.

As established in Lemma 2.2 and Lemma 2.3, the operations $\mathsf{parent}(T, v)$, $\mathsf{child}(T, v, i)$, and $\mathsf{degree}(T, v)$ are supported in $\mathrm{O}(1)$ time, while $\mathsf{access}(S, i)$, $\mathsf{rank}_v(S, i)$, and $\mathsf{select}_v(S, i)$ run in $\mathrm{O}(\lg n)$ time. Consequently, $\mathsf{N_{out}}(v, i)$, $\mathsf{N_{in}}(v, i)$, $\mathsf{degree}(v)$, and $\mathsf{adjacency}(u, v)$ can be computed in $\mathrm{O}(\lg n)$ time, while $\mathsf{N_{out}}(v)$ and $\mathsf{N_{in}}(v)$ require $\mathrm{O}(\lg n)$ time per neighbour. □

Towards the proof of Theorem 1.3, we now need to connect $H_0(A')$ and $H_0^{\deg}(S) = H_0(A)$. We start observing that the trivial bound for the empirical entropy, $H_0(A') \le (M-1)n \lg n$, yields the second term in Theorem 1.3–(b). Since this is also the expected value of our lower bound (computed by Łuczak et al. [LMS19a]), for typical graphs, our data structure uses asymptotically the optimal space for an unlabeled preferential-attachment graph.

For a closer analysis, it is convenient to consider the *per-character empirical entropy*: for a string $w \in \Sigma^n$ of length $n = |w|$, we define $H_0^{pc}(w) = \frac{1}{n} H_0(w) \le \lg |\Sigma|$. We point out that in general, deleting a character from a string may increase or decrease H_0^{pc}. In Appendix C, we will show that our scheme of choosing T, however, never increases the per-character entropy:

Lemma B.2: *Let G be any M-out-regular DAG G and let $A = A(G)$ and A' be as per our construction. Then $H_0^{pc}(A') \le H_0^{pc}(A)$.* ◁

Using Lemma B.2, the space from Lemma B.1 becomes

$$\begin{aligned} H_0(A') + 2n + o(Mn) &= H_0^{pc}(A')|A'| + 2n + o(Mn) \\ &= H_0^{pc}(A')(M-1)n + 2n + o(Mn) \\ &\le H_0^{pc}(A(S))(M-1)n + 2n + o(Mn) \\ &= H_0(A(S))\frac{(M-1)n}{Mn} + 2n + o(Mn) \\ &= H_0^{\deg}(S)(1 - \tfrac{1}{M}) + 2n + o(Mn). \end{aligned}$$

This concludes the proof of Theorem 1.3.

C. Proof of Lemma B.2 (Tree Deletion)

In this appendix, we prove Lemma B.2, the key lemma for bounding the space of the unlabelled graph data structure. We will first reformulate the problem as a question on strings, which we consider natural enough to be of independent interest.

C.1. Blocked Character Deletion

Let $X \cdot Y$ denote the concatenation of two strings X and Y, and let $X \setminus c$, for a character c, denote the result of deleting a single occurrence of c from X.

With that, we can translate the process by which A' results from A to strings: Given a string $A = X_1 \cdot X_2 \cdot \ldots \cdot X_n$ of n *blocks* (i.e. substrings) and alphabet Σ, we obtain string $A' = X_1' \cdot X_2' \cdot \ldots \cdot X_n'$ where, for $i \in [n]$, $X_i' = X_i \setminus c_i$, for $c_i \in X_i$; i.e., A' is obtained by deleting

exactly one character in each block of A. Can we achieve $H_0^{pc}(A') \leq H_0^{pc}(A)$? Our goal is to show that a simple greedy method, the LFC scheme below, suffices for that, provided all blocks are equally long, $|X_1| = |X_2| = \cdots = |X_n| = M$.

C.2. Least-Frequent-Character (LFC) Scheme

We will now describe an algorithm that corresponds to our construction of the tree T from Section 4.

First, let S be a sorted copy of A, where we sort characters by increasing frequency. Define the bijective function $\sigma : \Sigma \to [|\Sigma|]$ which maps each character of the alphabet Σ of A to a *rank* such that, for distinct characters $x, y \in \Sigma$, $|A|_x > |A|_y \implies \sigma(x) > \sigma(y)$ (such a mapping must exist). Then, construct the string S as follows: start with $S = \emptyset$, and for each i from 1 to $|\Sigma|$, append the character $\sigma^{-1}(i)$ $|A|_{\sigma^{-1}}$ times.

Example C.1: Let $n = 3$, $M = 4$, and $A = \texttt{abracadabraa}$. We can then define

$$\sigma = \{(\texttt{c}, 1), (\texttt{d}, 2), (\texttt{b}, 3), (\texttt{r}, 4), (\texttt{a}, 5)\},$$

and construct $S = \texttt{cdbbrraaaaaa}$. Note that there can be multiple ways to define σ. ◁

To construct A', we delete from each block its leftmost letter in S. Equivalently, but more convenient for the analysis to follow, assume without loss of generality that $\lambda \notin \Sigma$ and follow the procedure described in Algorithm 1.

Operation 1 Computation of A' following the LFC scheme.

$\hat{A} \leftarrow A$ ▷ *$\hat{A} = \hat{X}_1 \cdot \hat{X}_2 \cdot \ldots \cdot \hat{X}_n$ is a copy of $A = X_1 \cdot X_2 \cdot \ldots \cdot X_n$.*
$F[1..n] \leftarrow [0]^n$ ▷ *F stores which blocks in $\hat{A}$ were flagged*
for $i = 1$ to n **do**
  $k \leftarrow \min\{k' : S[k'] \neq \lambda\}$ ▷ *k is set to the smallest index such that $S[k] \neq \lambda$.*
  $c \leftarrow S[k]$ ▷ *c is set to the character at that index.*
  $j \leftarrow \min\{j' : c \in \hat{X}_{j'} \text{ and } F[j'] = 0\}$ ▷ *Index of some unflagged block where c occurs.*
  $F[j] \leftarrow 1$ ▷ *We register that the block has been flagged.*
  for all $l \in \hat{X}_j$ **do** ▷ *We iterate over all characters in the selected block $\hat{X}_j$.*
    $i' \leftarrow \min\{i'' : S[i''] = l\}$
    $S[i'] \leftarrow \lambda$
  $k \leftarrow \min\{k' : \hat{X}_j[k'] = c\}$
  $\hat{X}_j[k] \leftarrow \lambda$ ▷ *This also updates $\hat{A}$ since $\hat{X}_j \subseteq \hat{A}$.*
$A' \leftarrow \emptyset$
for $i \leftarrow 1$ to nM **do**
  if $\hat{A}[i] \neq \lambda$ **then**
    $A' \leftarrow A' \cdot \hat{A}[i]$ ▷ *A' is built by copying $\hat{A}$ and deleting positions where λ occurs.*
return A'

Initially, all blocks in A are unflagged. The scheme proceeds in n steps. In each step, it selects the leftmost non-λ character in S (call it c) located at position k (i.e. $S[k] = c \neq \lambda$), and flags every unflagged block that contains c; suppose it flags p such blocks. Then, in each of the p flagged blocks, it replaces exactly one occurrence of c in the block with λ. For each remaining character c' in the block, it replaces one occurrence of c' in S with λ. Finally, it replaces c itself with λ.

At each step, exactly M characters are replaced by λ in S, so that after n steps, all nM characters in S will have been replaced by λ. The string A' is obtained by deleting all occurrences of λ in A following the execution of the scheme.

Example C.2: Let $n = 3$, $M = 4$, and $A = \texttt{abracadabraa}$. Define σ as in Example C.1; we then get $S = \texttt{cdbbrraaaaaa}$. Let us compute A' following the scheme outlined in Algorithm 1.

# for	$\hat{A}$	S	F
	abracadabraa	cdbbrraaaaaa	$[0,0,0]$
1	abraλadabraa	λλbbrrλλaaaa	$[0,1,0]$
2	aλraλadabraa	λλλbrλλλλaa	$[1,1,0]$
3	aλraλadaλraa	λλλλλλλλλλλ	$[1,1,1]$

◁

Figure 3: Sample execution of the LFC scheme on the string $A = \texttt{abracadabraa}$. The resulting A' is araadaraa, and $H_0^{pc}(A') \approx 1.22439 \leq H_0^{pc}(A) \approx 1.95915$.

C.3. Analysis

The proof uses a handy tool to bound entropies: the *majorisation* partial order. Let $P = (p_1, p_2, \ldots, p_n)$ and $Q = (q_1, q_2, \ldots, q_n)$ be two n-sized distributions. We say that P *majorises* Q (i.e. $Q \preceq P$) if and only if $\sum_{i=1}^{k} p_i^{\downarrow} \geq \sum_{i=1}^{k} q_i^{\downarrow}$ for all $k \in [n]$, where $X^{\downarrow} = (x_1^{\downarrow}, x_2^{\downarrow}, \ldots, x_{|X|}^{\downarrow})$ denotes the $|X|$-sized vector of the distribution X sorted in non-increasing order. Given that the entropy function is *Schur-concave*, $Q \preceq P$ implies $H_0^{pc}(P) \leq H_0^{pc}(Q)$ [MOA79].

Assume $M \geq 2$; the case where $M = 1$ is trivial. Let S be a string over alphabet Σ, and let $P = (p_1, p_2, \ldots, p_{|\Sigma|})$ be a (non-ordered) distribution such that $p_i = \frac{|S|_{c_i}}{|S|}$ for all $i \in [|\Sigma|]$, where $c_i \in \Sigma$ is the ith character of the alphabet (without loss of generality, assume the latter is ordered).

Lemma C.3: *Let $P = (p_1, p_2, \ldots, p_n)$ and $Q = (q_1, q_2, \ldots, q_n)$ be two n-sized distributions such that $\sum_{i=1}^{k} p_i \geq \sum_{i=1}^{k} q_i^{\downarrow}$ for all $k \in [n]$. Then, $\sum_{i=1}^{k} p_i^{\downarrow} \geq \sum_{i=1}^{k} q_i^{\downarrow}$ for all $k \in [n]$.* ◁

Proof: Sorting P in non-increasing order does not reduce prefix sums, thus giving $\sum_{i=1}^{k} p_i^{\downarrow} \geq \sum_{i=1}^{k} p_i \geq \sum_{i=1}^{k} q_i^{\downarrow}$ for all $k \in [n]$. □

Lemma C.4: *Let $k_1 < k_2 < \cdots < k_n$ be the n indices in S picked by the LFC scheme (Algorithm 1, line 4). We have $k_i \leq (i-1)M + 1$.* ◁

Proof: Suppose, towards a contradiction, that at the ith selection, the leftmost non-λ character in S is at position $(i-1)M + 2$ or greater. This implies that for all $j \in [(i-1)M + 1]$, $S[j] = \lambda$.

Since $i - 1$ characters were selected previously, and $(M-1)(i-1)$ other characters were replaced by λ each time, then the next non-λ character must be at position at most $(M-1)(i-1) + (i-1) + 1 = M(i-1) + 1$. However, we assumed that position was occupied by λ, which is a contradiction. □

Lemma C.5: *Let A be a string of size nM and alphabet Σ, and construct its respective S following the LFC scheme. Let S' be obtained by copying S and simultaneously deleting its characters at positions $(i-1)M + 1$ for all $i \in [n]$. We then have $H_0^{pc}(S') \leq H_0^{pc}(S)$.* ◁

Proof: Let $f_1 \leq f_2 \leq \ldots f_{|\Sigma|}$ be the normalised frequencies of the characters of Σ in their order of appearance in S (by the way S is constructed, these must be non-decreasing), and let f_i' be the respective normalised frequency of the ith distinct character of S in S'; for instance,

suppose f_5 is the frequency of the fifth distinct character c in S, then f'_5 is the frequency of c in S' (i.e. after the n deletions in S).

The post-deletion frequency f'_i can be expressed in terms of f_i under two distinct cases:

$$f'_k = \begin{cases} \dfrac{nMf_i - \lceil nf_k \rceil}{n(M-1)}, & \text{(case 1) if } \left(n\sum_{i=1}^{k-1} f_i\right) \bmod M \;\geq\; M - (nf_k \bmod M) \\ \\ \dfrac{nMf_i - \lfloor nf_k \rfloor}{n(M-1)}, & \text{(case 2) otherwise.} \end{cases} \tag{6}$$

From those expressions, we want to show the following equality:

$$\sum_{i=1}^{k} f'_i = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{nM(M-1)}; \tag{7}$$

to do so, we proceed by induction.

Base case ($k = 1$). Clearly, the second case of Equation 6 applies for (one of) the largest character(s) (of frequency f_k in S), since there are no previous frequencies to add up, so $(n\sum_{i=1}^{0} f_i) \bmod M = 0 < M - (nf_1 \bmod M)$ (the RHS can never be 0). Hence, $f'_1 = \frac{Mnf_1 - \lfloor nf_1 \rfloor}{n(M-1)} \geq f_1$. The last inequality holds because:

$$\begin{aligned} \lfloor nf_1 \rfloor &\leq nf_1 \\ -M\lfloor nf_1 \rfloor &\geq -Mnf_1 \\ M^2nf_1 - M\lfloor nf_1 \rfloor &\geq M^2nf_1 - Mnf_1 \\ M(Mnf_1 - \lfloor nf_1 \rfloor) &\geq Mnf_1(M-1) \\ Mnf_1 - \lfloor nf_1 \rfloor &\geq nf_1(M-1) \\ \frac{Mnf_1 - \lfloor nf_1 \rfloor}{n(M-1)} &\geq \frac{nf_1(M-1)}{n(M-1)} \\ \frac{Mnf_1 - \lfloor nf_1 \rfloor}{n(M-1)} &\geq f_1. \end{aligned}$$

Induction hypothesis. Assume that

$$\sum_{i=1}^{k} f'_i = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn}$$

for some k. Let us show that the equality holds when adding f'_{k+1} in either of the two cases of Equation 6.

Case 1. We delete the character $\lceil nf_{k+1} \rceil$ times, i.e.

$$f'_{k+1} = \frac{Mnf_{k+1} - \lceil nf_{k+1} \rceil}{n(M-1)}.$$

Recall that this case occurs if and only if

$$\left(n\sum_{i=1}^{k} f_i\right) \bmod M \;\geq\; M - (nf_{k+1} \bmod M).$$

Therefore, we have

$$\frac{((n\sum_{i=1}^{k} f_i) \bmod M)}{n(M-1)} \geq \frac{M - (f_{k+1}n \bmod M)}{n(M-1)}$$

and since

$$M - (nf_{k+1} \bmod M) = M(\lceil nf_{k+1} \rceil - nf_{k+1}),$$

then

$$\frac{((n\sum_{i=1}^{k} f_i) \bmod M)}{n(M-1)} \geq \frac{M(\lceil nf_{k+1} \rceil - nf_{k+1})}{n(M-1)}.$$

Now, back to the assumed equality. Let us add the new frequency on both sides. We thus get

$$\sum_{i=1}^{k} f'_i + f'_{k+1} = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + f'_{k+1},$$

hence giving us

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{Mnf_{k+1} - \lceil nf_{k+1} \rceil}{n(M-1)}.$$

By splitting the fraction, we get:

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{nf_{k+1} - \lceil nf_{k+1} \rceil}{n(M-1)} + \frac{n(M-1)f_{k+1}}{n(M-1)}$$

which gives us, by cancelling out the factors in the last fraction,

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{nf_{k+1} - \lceil nf_{k+1} \rceil}{n(M-1)} + f_{k+1}$$

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{nf_{k+1} - \lceil nf_{k+1} \rceil}{n(M-1)}.$$

By multiplying the numerator and denominator by M in the last summand, we obtain

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{M(nf_{k+1} - \lceil nf_{k+1} \rceil)}{Mn(M-1)}$$

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} - \frac{M(\lceil nf_{k+1} \rceil - nf_{k+1})}{Mn(M-1)}.$$

Note that we can already establish that $\sum_{i=1}^{k+1} f'_i \geq \sum_{i=1}^{k+1} f_i$ since

$$\frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} \geq \frac{M(\lceil nf_{k+1} \rceil - nf_{k+1})}{Mn(M-1)} \geq 0.$$

Finally, we have

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} - \frac{M - (nf_{k+1} \bmod M)}{Mn(M-1)}$$

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k+1} f_i) \bmod M}{(M-1)Mn},$$

where the last equality holds because

$$\left(n\sum_{i=1}^{k+1} f_i\right) \bmod M + (nf_{k+1}) \bmod M \geq M,$$

which strictly follows from the condition of the case.

Case 2. We delete the character $\lfloor nf_{k+1} \rfloor$ times, i.e.

$$f'_{k+1} = \frac{Mnf_{k+1} - \lfloor nf_{k+1} \rfloor}{n(M-1)}.$$

Again, let us add the new frequency on both sides. We have

$$\sum_{i=1}^{k} f'_i + f'_{k+1} = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + f'_{k+1}$$
$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{Mnf_{k+1} - \lfloor nf_{k+1} \rfloor}{n(M-1)}.$$

By splitting the fraction, we get

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{nf_{k+1} - \lfloor nf_{k+1} \rfloor}{n(M-1)} + \frac{n(M-1)f_{k+1}}{n(M-1)}$$
$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{nf_{k+1} - \lfloor nf_{k+1} \rfloor}{n(M-1)} + f_{k+1}$$
$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{nf_{k+1} - \lfloor nf_{k+1} \rfloor}{n(M-1)}.$$

By multiplying the numerator and denominator by M in the last summand, we obtain

$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn} + \frac{M(nf_{k+1} - \lfloor nf_{k+1} \rfloor)}{Mn(M-1)}$$
$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M + M(nf_{k+1} - \lfloor nf_{k+1} \rfloor)}{(M-1)Mn}$$
$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k} f_i) \bmod M + nf_{k+1} \bmod M}{(M-1)Mn}$$
$$\sum_{i=1}^{k+1} f'_i = \sum_{i=1}^{k+1} f_i + \frac{(n\sum_{i=1}^{k+1} f_i) \bmod M}{(M-1)Mn},$$

where the last equation strictly follows from the condition of the case, i.e.

$$\left(n\sum_{i=1}^{k+1} f_i\right) \bmod M + (nf_{k+1}) \bmod M < M.$$

In either case, the equality holds. Notice that we have, for all $k \in [|\Sigma|]$:

$$\sum_{i=1}^{k} f'_i \geq \sum_{i=1}^{k} f_i$$

since $\frac{(n\sum_{i=1}^{k} f_i) \bmod M}{(M-1)Mn}$ is non-negative.

Let $P = (f_1, f_2, \ldots, f_{|\Sigma|})$ and $P' = (f'_1, f'_2, \ldots, f'_{|\Sigma|})$. Clearly, by Lemma C.3 (note that $P = P^{\downarrow}$), the last inequality implies that $P^{\downarrow} \preceq P'^{\downarrow}$. This further implies that $H_0^{pc}(P'^{\downarrow}) \leq H_0^{pc}(P^{\downarrow})$, and the lemma follows. □

Let $k_1 < k_2 < \cdots < k_n$ be the indices picked by the LFC scheme. Given that $k_i \leq (i-1)M+1$ for $i \in [n]$, then any set of indices from the scheme can be obtained by setting all the indices to their respective upper bounds and *shifting* them to the left (i.e. decreasing their respective k).

Lemma C.6: *Let $k_1 < k_2 < \cdots < k_n$ be the indices selected by the LFC scheme and let*

$$\sum_{i=1}^{j} f'_i \geq \sum_{i=1}^{j} f_i$$

for all $j \in [n]$. If some k_i is set to $k_i - x$, with $x \in \mathbb{N}$, such that $k_i \geq 1$ and $k_i \neq k_j$ for all $j \in [n]$, then the inequality still holds. ◁

Proof: Suppose $S[k_i] = S[k_i - x]$. Clearly, *shifting* the index to the left does not change the frequency of any character in Σ; therefore, in that case, the inequality will still hold.

Now, suppose $S[k_i] \neq S[k_i - x]$ and let f'_a and f'_b be the impacted frequencies, with $a < b$. Shifting the index is equivalent to restoring the character at position k_i and deleting the character at position $k_i - x$. Thus, f'_b decreases by some value δ, and f'_a increases by δ. This clearly does not affect the prefix sums, and the lemma follows. □

Proof of Lemma B.2: Construct S, initially set the "deleting" indices to $1, M+1, \ldots, (n-1)M+1$, and let S' be the string obtained by deleting the characters of S at those indices. By Lemma C.5, $H_0^{pc}(S') \leq H_0^{pc}(S)$. We know that any set of indices computed by the scheme on A must result with indices $k_i \leq (i-1)M + 1$; so, for each index k_i, shift them to wherever the LFC scheme would put them on input A. That shifting still does not increase the entropy beyond that of S, by Lemma C.6; thus $H_0^{pc}(S') \leq H_0^{pc}(S)$ holds after the shifts.

Since $H_0^{pc}(A) = H_0^{pc}(S)$, $H_0^{pc}(S') = H_0^{pc}(A')$ and $H_0^{pc}(S') \leq H_0^{pc}(S)$, it follows that $H_0^{pc}(A') \leq H_0^{pc}(A)$, completing the proof. □

Remark C.7 (Equal-sized blocks needed): We point out that, in contrast to Lemma B.2, allowing blocks of *different* lengths in blocked character deletion can make it impossible to achieve $H_0^{pc}(A') \leq H_0^{pc}(A)$. ◁