Rooting Out Entropy: Optimal Tree Extraction for Ultra-Succinct Graphs

Ziad Ismaili Alaoui* Tamio-Vesa Nakajima† Namrata‡
Sebastian Wild§

March 15, 2026

Abstract

We combine two methods for the lossless compression of unlabeled graphs – entropy compressing adjacency lists and computing canonical names for vertices – and solve an ensuing novel optimisation problem: Minimum-Entropy Tree-Extraction (MINETREX). MINETREX asks to determine a spanning forest F to remove from a graph G so that the remaining graph $G - F$ has minimal indegree entropy $H(d_1, \dots, d_n) = \sum_{v \in V} d_v \log_2(m/d_v)$ among all choices for F. (Here d_v is the indegree of vertex v in $G - F$; m is the number of edges.)

We show that MINETREX is NP-hard to approximate with additive error better than δn (for some constant $\delta > 0$), and provide a simple greedy algorithm that achieves additive error at most $n/\ln 2$. By storing the extracted spanning forest and the remaining edges separately, we obtain a degree-entropy compressed ("ultrasuccinct") data structure for representing an arbitrary (static) unlabeled graph that supports navigational graph queries in logarithmic time. It serves as a drop-in replacement for adjacency-list representations using substantially less space for most graphs; we precisely quantify these savings in terms of the maximal subgraph density.

Our inapproximability result uses an approximate variant of the hitting set problem on *biregular* instances whose hardness proof is contained implicitly in a reduction by Guruswami and Trevisan (APPROX/RANDOM 2005); we consider the unearthing of this reduction partner of independent interest with further likely uses in hardness of approximation.

1. Introduction

In many applications of network analysis, only the *structure* of the underlying graph is of importance – the names or labels of most vertices are often irrelevant. With growing data sizes, it is clearly desirable not to waste space on storing such irrelevant names; on "edge devices" it may even become imperative to work on compressed representations of data. The thriving area of space-efficient data structures – and specifically succinct graph data structures – is testimony to this desire.

However, when "writing down" a general graph (not known to be from a restricted class of graphs), it seems inevitable that we use *some* vertex identifiers by which we can refer to a

*University of Liverpool, United Kingdom, ziad.ismaili-alaoui@liverpool.ac.uk
†University of Marburg, Germany, nakajima@informatik.uni-marburg.de
‡University of Birmingham, United Kingdom, n.namrata@bham.ac.uk
§University of Marburg, Germany, wild@informatik.uni-marburg.de

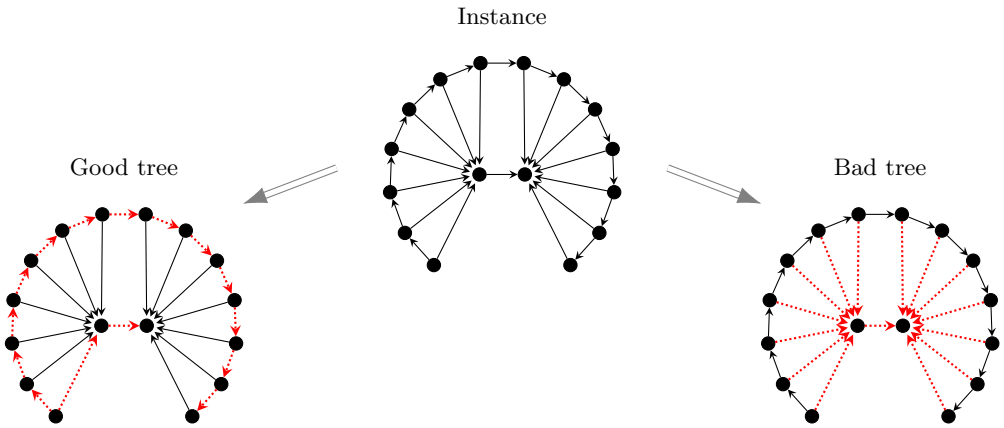

Figure 1: Example showing the significance of the tree in Minimum-Entropy Tree-Extraction. In the example graph (**middle**), tree extraction can, depending on the chosen tree (red dotted), either (**left**) yield the optimal space saving of $\sim n \lg n$ bits for the entropy-compressed representation of the remaining edges (black), or (**right**) negligible saving of $O(n)$ bits, which is comparable to its overhead to store the extracted tree. We point out that the directions of edges in the extracted tree can be arbitrary; tree extraction is not limited to deleting an arborescence.

vertex when describing the edges of the graph. And even if we obtain these vertex ids as some canonical names derived solely from the graph's topology, classical graph representations such as adjacency lists *already* encode vertex ids implicitly (the *order* of vertices in the sequential representation). So is it even thinkable to save the space for useless labels *without further assumptions on the stored graphs?*

We give a resounding *yes* to this question: Our *"tree-extraction"* data structure is (1) simple enough to have practical appeal, (2) often (under conditions we analyze below) achieves an asymptotically optimal space reduction at (3) the expense of a very modest slowdown for operations and construction time. *Tree Extraction (TREX)* (Figure 1) combines two methods for the lossless compression of graphs – entropy compressing adjacency lists and computing canonical names for vertices. This combination gives rise to a novel optimisation problem, whose study and exploitation is the main technical contribution of this paper: Minimum-Entropy Tree-Extraction (MINETREX).

Labelled Graph Representations. For ease of presentation, let $G = (V, E)$ be a *directed* and connected[1] graph; we generalise our results to undirected graphs later. We assume $V = [n] = \{1, \ldots, n\}$. For $v \in V$, denote by $d_v = d_v^{\text{in}}$ the *indegree* of v, i.e., the number of incoming edges $(w, v) \in E$. A textbook adjacency-list representation of G using linked lists uses $\Theta(m \log n)$ bits of space when G has $n = |V|$ vertices and $m = |E| \geq n - 1$ edges. A more careful array-based representation using $m\lceil \lg n \rceil + n\lceil \lg m \rceil$ bits of space[2] is easily possible: store all adjacency lists concatenated in one long array $A[1..m]$, plus an array $S[1..n]$ for the starting indices so that $N^+(v)$, the outneighbourhood of v is found in $A[S[v]..S[v] + d_v^{\text{out}})$. By replacing the array A of integers in $[1..n] = V$ with a *wavelet tree* [GGV03] (see also Lemma 4.2), we obtain an entropy-compressed representation of G [Nav14]: The total space

[1] Connectivity simplifies description and analysis of our data structure; our ideas can easily be adapted for disconnected graphs, too.

[2] Here and throughout, we write lg for $\log_2$ and assume the word-RAM model with word size $\Theta(\log n)$.

becomes $m\mathcal{H}^{\text{in}}_{\deg}(G) + o(m) + n\lceil \lg m \rceil$ bits, where

$$\mathcal{H}^{\text{in}}_{\deg}(G) \;=\; \sum_{v=1}^{n} \frac{d^{\text{in}}_v}{m} \lg\!\left(\frac{m}{d^{\text{in}}_v}\right)$$

denotes the (zero-th order) empirical *indegree entropy* of G.[3] As a bonus, with the wavelet tree's rank/select operations, we can navigating over *incoming* edges [Nav14]; all operations take $O(\lg n)$ time.

Since $\mathcal{H}^{\text{in}}_{\deg}$ coincides with the empirical entropy of the *adjacency string* $A[1] \dots A[m]$ of edge targets, it is a natural instance-specific measure of compressibility of G. It is also the information-theoretic instance-specific lower bound in natural models of random graphs: $m\mathcal{H}^{\text{in}}_{\deg}(G) \sim \lg(1/\mathbb{P}[G])$ for $\mathbb{P}[G]$ the probability of G to arise according to the *directed configuration model* (fixed degree sequence model) and the (labelled) Barabási-Albert model of preferential attachment graphs.[4] In both cases, *for the labelled graph with given vertex ids*, a simple wavelet-tree based representation uses *instance-optimal* space.

Tree Extraction. We explore how to extend the above results to *unlabelled* graphs. The key idea of *Tree Extraction (TREX)* is to store some of the graph edges as part of a (rooted, ordered) *tree* T, whose topology can be stored with 3 bits per edge (2 for the tree shape, 1 for the edge orientation) – much fewer than, say, the $\lg n$ bits in the adjacency string A. We emphasise that we select an undirected spanning tree, not an arborescence; the edges need not be oriented towards the root. In other words, we select a subset of edges which, when ignoring edge orientations, form a spanning tree. Storing the shape of T using $\sim 2n$ bits of space can be achieved using any of the many succinct tree data structures, e.g., [Jac89, MR01, GRR06, NS14, BDM$^+$05, JSS12, FM14, HMN$^+$20].

Normally, such savings are entirely moot, since we have to also store which tree node corresponds to which graph vertex – adding back $\lg(n!) \sim n \lg n$ bits of space. But when we are *not interested* in the original vertex ids, we might as well let T choose ids of its personal convenience – *canonical names* induced by the tree data structure – and use these names for storing the $m-n+1$ remaining edges in $E(G) \setminus E(T)$. For the simple array-based adjacency lists with $\lceil \lg n \rceil$ bits per edge, tree extraction *always* yields space savings of $\sim n \lg n$ bits irrespective of how we choose T,[5] but for the entropy-compressed wavelet-tree representation, the tree can make a huge difference (Figure 1). *How should we choose the tree to extract? Can we obtain an instance-optimal unlabelled graph representation?*

Minimum-Entropy Tree-Extraction. This first question is precisely the Minimum-Entropy Tree-Extraction (MINETREX) problem: given a directed, connected graph G, find a spanning tree T that minimises $\mathcal{H}^{\text{in}}_{\deg}(G - T)$. (In principle, we could use any spanning *forest*, but there always exists a spanning tree that is an optimal solution, cf. Appendix B.)

To our knowledge, this combinatorial optimisation problem has never been studied. We show that MINETREX is NP-hard, and indeed, that there exists a constant $\delta > 0$ such that it is NP-hard to even find a solution of cost $\leq OPT + \delta n$. The former is by a simple reduction from Exact Cover by 3-Regular 3-Sets (X3C); the latter extends this reduction to work with a biregular approximate variant of Exact Hitting Set (EHS) ((β, r, k)-Almost-EHS). Even though the hardness of this problem is implicitly contained in a known result [GT05], to

[3] Here and throughout, we take the standard convention that $x \lg x = x \lg(1/x) = 0$ for $x = 0$.

[4] Here and throughout, $f(n) \sim g(n)$ iff $\lim f(n)/g(n) = 1$.

[5] This explains why the (to our knowledge) first (implicit) use of tree extraction – in the GLOUDS data structure – simply hard-coded the use of a BFS tree [FP16].

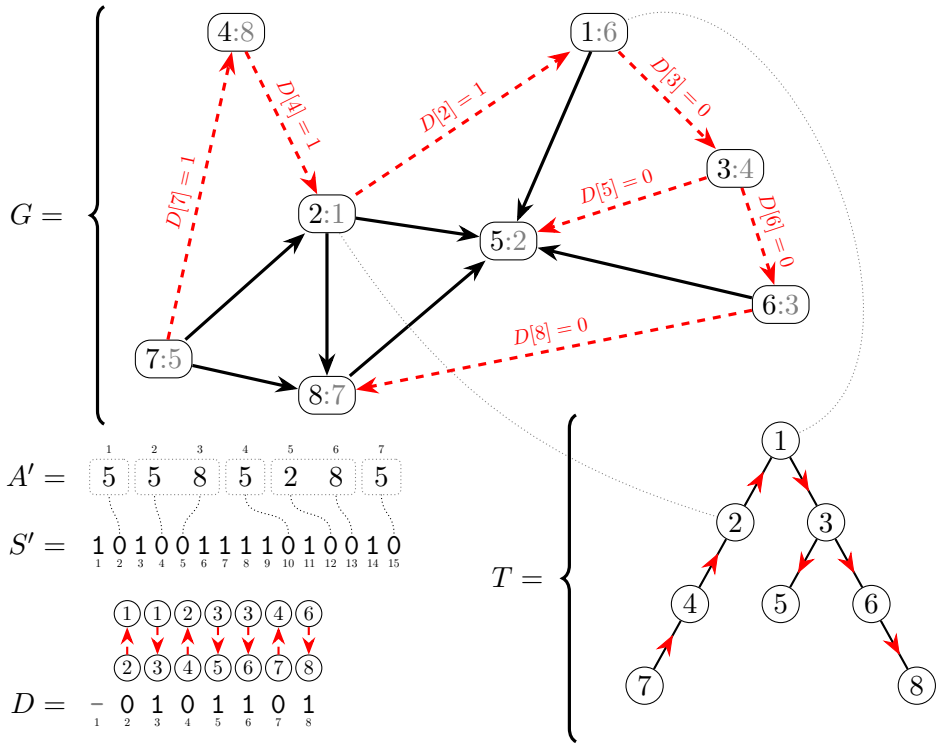

Figure 2: Illustration of the ultrasuccinct TREX data structure on a directed graph G. The dashed edges form the spanning tree T (computed via the MST-approximation algorithm from Section 2). The vertices are labelled as 'new:old' where the new labels are induced by the level-order traversal of rooted tree T. The array D stores the direction of edges in T. The array A' represents the adjacency list of G after removing T in the order of vertices appearing in level-order. The one-bits in S' mark the start indices of the outneighbourhood of vertices in A'.

our knowledge, the explicit formulation of this latter gap problem is novel, which we consider of potential interest for other inapproximability reductions, e.g. turning X3C reductions into hardness-of-approximation reductions.

On the positive side, we show that a natural greedy algorithm works exceedingly well. Any edge included in the tree does not have to be stored in the wavelet tree. Ignoring the effect that its very removal has on the resulting empirical entropy, extracting an edge (u, v) saves us $\lg(m/d_v) = \lg m - \lg d_v$ bits. A greedy approach would thus sort the edges by decreasing savings $\lg(m/d_v)$ – or equivalently, by increasing target degree d_v – and select an optimal spanning tree using these edge weights. Despite being both myopic and ignorant to its own impact, we prove that this greedy MST approach finds a spanning tree T of MINETREX cost $\leq OPT + n/\ln 2$, and is thus, w.r.t. its approximation guarantee, optimal up to constants (among all polytime algorithms, unless P = NP).

TREX and Twin Removal Data Structure. We combine this data structure with *twin removal*: u and v are twins if their neighbourhoods coincide; for an unlabelled graph, it suffices to remember the number of twins of each vertex instead of storing u and v separately. TREX with twin removal yields an instance-optimal-space data structure for unlabelled graphs arising from the *random copy model* [TMS20] (see also Appendix C).

Structure Over Identity. We point out that the relabeling mapping used in tree extraction can be explicitly output at construction time, so a user of the data structure may choose to remember some of the new vertex labels to facilitate later queries. However, the mapping is not stored as part of our TREX data structure. We are thus storing an *unlabelled graph*, i.e., only the equivalence class w.r.t. graph isomorphism instead of a concrete labelled graph. This reduces the intrinsic information content by up to $\lg(n!) \sim n \lg n$ bits, and these maximal savings are asymptotically attained for various distributions over graphs [CS12, LMS19, PHS+22, INW25].

Apart from the information-theoretic question of identifying the intrinsic information content of graph-structured data [SG18] (in particular the notion of *structural entropy*), many applications can profit from these space savings. For example, identifying *network motifs*, i.e., small subgraphs that occur surprisingly often or surprisingly rarely in a network compared to random graphs, is a standard tool in network analysis [MSOI+02]. This involves counting occurrences of many motifs in many random graph chosen from a null model. Here, vertex ids are randomly generated anyways, so renaming vertices does not change the result.

To further stress how widespread working with unlabelled graphs is, we point out that the area of succinct graph data structures is full of examples where original labels are not stored. Indeed, preserving the original vertex names would often dominate overall space (e.g. in trees [NS14, FM14], planar graphs [Jac89, FFSG+20, KM23], proper interval graphs [ACJRS20, HMN+20], bipartite permutation graphs [TWZ23], bounded clique-width graphs [Kam17, CJSRS24], separable graphs [BF10]) or affect the leading term of space usage (e.g. in interval graphs [ACJRS20], permutation graphs [TWZ23] circle graphs [ACJ+22]). In all these cases, the known succinct data structures store unlabelled graphs, often leaving the assumption implicit that renaming vertices must be acceptable.

Undirected Graphs. If we start with an undirected graph, we have even more freedom: We may extract a spanning tree for assigning vertex names, and also orient all edges to minimise the resulting indegree entropy. (Recall that wavelet trees already support navigation over both outgoing and incoming edges, so storing the edge in one orientation is sufficient.) Both our hardness results and our approximation algorithms directly generalise to this case.

1.1. Related Work

Compressed representations of graphs are in strong demand and corresponding data structures are an active area of research. Much work in this area falls into one of two camps:

(1) Applied graph compression. Here, with properties of certain domains in mind, compression heuristics are developed and evaluated empirically on benchmarks datasets. A representative example is the webgraph framework [BV04], which provides space-efficient data structures for typical networks of websites and hyperlinks. Many more are discussed in [BH19].

(2) Succinct representation of a graph class. A graph class $\mathcal{G}$ is a set of graphs closed under isomorphism. A succinct representation of a graph class uses $\lg |G_n|(1+o(1))$ bits of space, where G_n is the set of graphs with vertex set $[n]$ in $\mathcal{G}$. In other words, we asymptotically use the optimal worst-case number of bits to represent graphs with a known number of vertices n. Succinct representations with efficient (often $O(1)$ time) navigational query support are known for many graph classes; as nonexhaustive examples, we list (proper) interval graphs [ACJRS20, HMN+20], (bipartite) permutation graphs [TWZ23], bounded clique-width graphs [Kam17, CJSRS24], separable graphs [BF10], circle graphs [ACJ+22], chordal graphs [MW18]. Planar maps can also be succinctly represented [CADS08]; planar *graphs* (without a fixed embedding) still pose challenges since not even the asymptotic number of unlabelled graphs on n vertices is known,

but representations with $O(\lg |G_n|)$ bits are achievable [FFSG$^+$20, KM23]. All of the above works (implicitly) consider unlabelled graphs and assign new vertex ids at construction time.

A common property of the above succinct representations of a graph class is that they use the *same* space for any graph of a given size, and are highly specific to the graph class at hand. In general, 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 $\mathbb{P}[x]$. A much smaller number of prior works consider data structures whose space *adapts* to the compressibility of the concrete instance; such data structures are known for trees [JSS12, MNSBW21] and preferential-attachment graphs [INW25].

We generalise the work of [INW25] from the specific model of out-regular graphs generated by preferential attachment *(Barabási-Albert model)* to arbitrary directed or undirected graphs. Unlike in [INW25], general graphs require a principled choice of the spanning tree to extract and a more elaborate analysis. Especially for undirected graphs, we also consider the additional degree of freedom to choose an orientation for all edges. Like in [INW25], our resulting algorithms are simple and use only standard tools from succinct data structures, for which efficient implementations are available.

GLOUDS [FP16] is a precursor to [INW25] and similar to our data structure in that they also partition the graph into a tree and the remaining edges. However, the focus there is on a simple data structure. The extracted tree is always a BFS tree/forest and the remaining edges are encoded *alongside* the LOUDS (level-order unary degree sequence) representation of the tree. Non-tree edges are only compressed in that their target is stored using $\lceil \lg h \rceil$ bits where h is the number of different vertices that appear as targets of the $k = m - (n - 1)$ non-tree edges; however, no attempt is made at choosing a tree to obtain a minimal h. Storing edge targets in an entropy-compressed form is not pursued further.

The problem of orienting edges of an undirected graph to minimise the resulting indegree entropy (without tree extraction) was considered by Cardinal et al. [CFJ08]. They showed that the problem is NP-hard even if the graph is planar. On the other hand, simply orienting all edges towards the larger-degree endpoint yields an indegree entropy of at most $OPT + m$. We can prove a similar result, but with weaker constants, via Theorem 1.1.

1.2. Contributions

In this section, we succinctly formalise our technical contributions. Proofs are delayed to later sections.

Our first result is a generic way to obtain approximate algorithms for *entropy minimisation* by transforming exact algorithms for *linear minimisation*. (This theorem may seem abstract at first, however it immediately implies that several of the problems we are interested can be approximated up to a linear error term; we illustrate this below on MINETREX.)

Theorem 1.1: *Fix some vector $(d_1, \ldots, d_n) \in \mathbb{N}^n$ with $m = \sum_i d_i$. Fix some positive integer k. Fix also some class*

$$\mathcal{C} \subseteq \left\{ (x_1, \ldots, x_n) \in \mathbb{N} \;\middle|\; \forall i \; 0 \le x_i \le d_n \;\wedge\; \sum_i x_i = k \right\}.$$

Consider the following two objective functions:

$$\textsc{Opt-Lin}(x_1, \ldots, x_n) \;=\; \sum_i x_i \lg d_i,$$

$$\textsc{Opt-Ent}(x_1,\ldots,x_n) \;=\; -\sum_i (d_i - x_i)\lg\!\left(\frac{d_i - x_i}{m-k}\right) \;=\; H(d_1 - x_1,\ldots,d_n - x_n).$$

The following holds for any $(x_1,\ldots,x_n) \in \mathcal{C}$*:*

$$\begin{aligned}
&\textsc{Opt-Ent}(x_1,\ldots,x_n) \;-\; \frac{k}{\ln 2} \\
&\leq\; (m-k)\lg(m-k) - \left(\sum_i d_i \lg d_i\right) \;+\; \textsc{Opt-Lin}(x_1,\ldots,x_n) \\
&\leq\; \textsc{Opt-Ent}(x_1,\ldots,x_n).
\end{aligned}$$

Hence, if we can minimise Opt-Lin *over* $\mathcal{C}$ *exactly, then we can minimise* Opt-Ent *over* $\mathcal{C}$ *with additive error at most* $k/\ln 2$*.* ◁

Let us see how to apply this theorem to MINETREX. Suppose we are given a (weakly) connected digraph G with n vertices and m edges. Let $d_1,\ldots,d_n$ be the indegree sequence of G. Take $k = n-1$, and define $\mathcal{C}$ to be the set of indegree sequences of spanning trees of G.[6] In other words, $\mathcal{C}$ contains, for every spanning tree T of G, the sequence $x^{(T)} = (x_1^{(T)},\ldots,x_n^{(T)})$ where

$$x_i^{(T)} \;=\; \big|\{(j,i) \in T\}\big|.$$

Notice that for any spanning tree T of G, we have that the empirical entropy of the indegree sequence of $G - T$ is, by definition, $\textsc{Opt-Ent}(x^{(T)})$. Hence, the MINETREX problem is identical to minimising Opt-Ent over $\mathcal{C}$.[7] In view of Theorem 1.1, this can be done with error $(n-1)/\ln 2$, provided we can minimise Opt-Lin over $\mathcal{C}$. But, expanding definitions, this is just finding a minimum cost spanning tree where the cost of edge (i,j) is given by $\lg d_j$. This can be done easily – hence the following key corollary follows.

Corollary 1.2: *MINETREX can be approximated with additive error* $(n-1)/\ln 2$ *in* $O(n+m)$ *time.* ◁

Proof: The previous discussion implies that it suffices to compute a minimum cost spanning tree where the cost of an edge (i,j) is $\lg(d_j)$, where d_j is the indegree of j. To do this in linear time, observe that we can just as well assign edge (i,j) cost d_j without modifying the minimum cost spanning tree. Then, since the new costs are integers between 1 and m, we can easily find the desired spanning tree using Prim's algorithm in $O(n+m)$ time with a bucket-based priority queue. □

In a similar way (see Section 2 for proof), we can deal with selecting both the orientation of edges on top of tree extraction. We denote this problem as Undirected Minimum-Entropy Tree-Extraction (U-MINETREX).

Corollary 1.3: *U-MINETREX can be approximated with additive error* $(m+n-1)/\ln 2$ *in* $O(n+m)$ *time.* ◁

Let us now move on to hardness. We prove that for polytime algorithms, we will have to content ourselves with a linear error term unless $\mathsf{P} = \mathsf{NP}$, thus resolving the approximability of (U-)MINETREX up to the first order.

[6] To clarify: we are interested in finding subsets of edges that are trees when ignoring edge orientations; this is what is meant by spanning tree.

[7] Strictly speaking, we need to return T not the vector $x^{(T)}$, but this is also easy to compute.

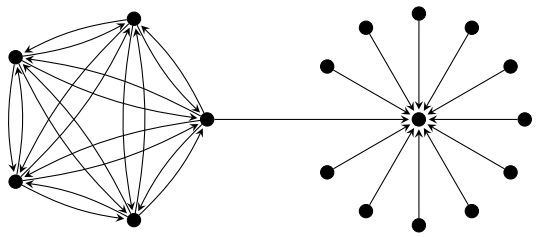

Figure 3: Example of a graph where tree extraction does not lower entropy. The clique on the left has $\sqrt{n}$ vertices in general.

Theorem 1.4: *There exists some $\delta > 0$ such that it is NP-hard to approximate MINETREX within additive error δn.* ◁

Theorem 1.5: *There exists some $\delta > 0$ such that it is NP-hard to approximate U-MINETREX within additive error δm.* ◁

These theorems follow by a reduction from a hardness result by Guruswami and Trevisan [GT05] on Gap Exact Hitting Set.

While we find Minimum-Entropy Tree-Extraction interesting in its own right, from the application perspective of compressing graphs, finding the best tree to extract would be utterly useless if either (i) every tree is more or less equally good, or (ii) *no* tree leads to a significant savings whatsoever.

It is easy to see that (i) is not true in general. Indeed, consider Figure 1, or more generally a graph with $2n + 2$ vertices, labelled $a_1, \dots, a_n$, $b_1, \dots, b_n$, x, y, with edges $a_1 \to \cdots \to a_n \to b_1 \to \cdots \to b_n$, $a_i \to x$, $b_i \to y$ and $x \to y$. In such a graph, if we remove the path

$$y \leftarrow x \leftarrow a_1 \to \cdots \to a_n \to b_1 \to \cdots \to b_n,$$

we are left with the edges $a_2 \to x, \dots, a_n \to x$ and $b_1 \to y, \dots, b_n \to y$ (two disjoint stars). Our scheme stores the resulting graph using $O(n)$ bits (using 1 bit per edge in $G - T$, for a total of $\sim 2n$ bits plus neighbourhood boundaries for $G - T$). On the other hand, if we remove the edges $a_1 \to x, \dots, a_n \to x$, $b_1 \to y, \dots, b_n \to y$ and $x \to y$, then we are left with the path $a_1 \to \cdots \to a_n \to b_1 \to \cdots \to b_n$, with indegree entropy $\Omega(\lg n)$ bit (as for the original graph if we do not remove anything). Thus we can achieve a significant space saving using our approximation scheme – and indeed, the error of $O(n)$ is negligible compared to that saving.

Question (ii) is rather more thorny. It turns out that there are cases where *no* tree significantly lowers the entropy if deleted. Consider, for example, a graph that looks like the graph from Figure 3. The general pattern shall have a clique on the left with k vertices and a star on the right with k^2 vertices; so $k \sim \sqrt{n}$. In this example, the original graph has $\mathcal{H}^{\text{in}}_{\deg}(G) \sim \frac{1}{2} \lg k \sim \frac{1}{4} \lg n$, for a total of $\Theta(n \lg n)$ bits to store G. The problem is that the star on the right side of the graph costs almost nothing to store – most of the complexity of the graph is contained in the small but dense part on the left. This dense part is also almost completely untouched by removing any tree – we see that, indeed, any tree T that we remove only affects a vanishingly small fraction of the edges in the clique, leaving $\mathcal{H}^{\text{in}}_{\deg}(G - T) \sim \frac{1}{4} \lg n$.

The essential reason why Figure 3 has this bad behaviour is that its density is completely contained within a small part: the clique was much denser than the whole graph. This is problematic because the tree cannot lower the amount of bits needed to store the adjacency information in the dense part of the graph very much. On the other hand, the vast majority of the graph is nearly free to store, and thus we do not profit much from removing it. We show that this is essentially the *only* case where this happens: for any digraph G where no subgraph

is significantly denser than the entire graph, we show that we can always achieve a significant saving via tree extraction.

In the following, for a graph G with n vertices and m edges, define its density $\delta(G) = m/n$.

Theorem 1.6: *Consider some connected n-vertex m-edge graph $G = (V, E)$. Let α be some constant so that for any subgraph G' of G, we have $\delta(G') \leq \alpha\delta(G)$. Then, there exists some spanning tree T of G such that*

$$H(G - T) \;\leq\; H(G) - \frac{n}{2\alpha}\mathcal{H}^{\text{in}}_{\text{deg}}(G) + \frac{2n}{\ln 2}. \qquad \triangleleft$$

For example, if G has per-edge entropy say $\frac{1}{3}\lg n$, and no subgraph has *proportionally* more than 2 times the number of edges as the entire graph, then we can save at least roughly $\frac{1}{12}n\lg n - 2n/\ln 2$ bits. While we make no claim about whether the bound in Theorem 1.6 is tight, note that if the graph has, e.g., constant per-edge entropy $\mathcal{H}^{\text{in}}_{\text{deg}}$, there is no hope to save more than a linear number of bits.

Additionally, we show that in any digraph where a significant number of vertices have nonzero indegree, we also achieve a significant savings.

Theorem 1.7: *Consider some connected n-vertex m-edge graph $G = (V, E)$, where at least n/α vertices have nonzero indegree. Then there exists some spanning tree T of G such that*

$$H(G - T) \;\leq\; H(G) - \frac{n}{2\alpha}\mathcal{H}^{\text{in}}_{\text{deg}}(G) + \frac{2n}{\ln 2}. \qquad \triangleleft$$

For example, if G has per-edge entropy say $\frac{1}{3}\lg n$, and at least $1/3$ of the vertices have nonzero indegree, then we can save roughly $\frac{1}{18}n\lg n - 2n/\ln 2$ bits.

Finally, we provide a data structure capable of storing a TREXed unlabelled graph while supporting all operations a classical adjacency list representation offers (plus navigation to inneighbours) in $O(\lg n)$ time (see also Figure 2).

Theorem 1.8 (Ultrasuccinct TREX): *Given a directed graph G and spanning tree T in G, we can construct a data structure for a graph isomorphic to G (where the vertex relabeling can be output at construction time) that occupies $H(G - T) + 3n + o(m) + \lg\binom{m}{n} \leq H(G - T) + n\lg(m/n) + 4.4427n + o(m)$ bits of space, while supporting the operations $\mathsf{N}_{\mathsf{out}}(v, i)$ (ith outneighbour of v), $\mathsf{N}_{\mathsf{in}}(v, i)$ (ith inneighbour of v), $\mathsf{indegree}(v)$, and $\mathsf{adjacent}(u, v)$ in $O(\lg n)$ time; $\mathsf{outdegree}(v)$ takes $O(1)$ time.* $\triangleleft$

Lastly, we show how to combine the above data structure with *twin removal*, a further compression method for unlabelled graphs where we identify all vertices with identical neighbourhood. Details are given in Section 5.

1.3. Hardness of exactly solving MINETREX.

While we defer the proof of Theorem 1.4 and Theorem 1.5 to Section 3, to get a taste of the reduction, we informally sketch the key idea for the proof of the simpler statement that *exact* MINETREX is NP-hard.

Recall that in the Exact Cover By 3-Regular 3-Sets (X3C) problem, we are given a universe $\{1, \ldots, 3n\}$, and a family of sets $S_1, \ldots, S_{3n} \subseteq [3n]$, such that every set has size 3, and every element belongs to exactly 3 sets. The goal is then to select n sets so that every element in the universe is contained within precisely one of the selected sets, or to determine that this is impossible.

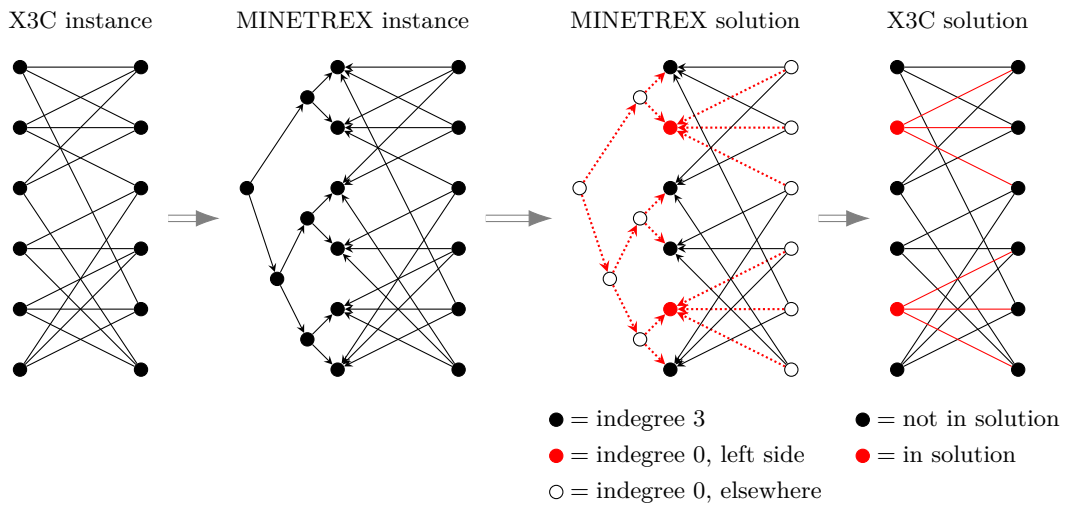

Figure 4: Reduction of X3C to MINETREX (the left-hand-side bipartition represents the subcollections, and the right-hand-side bipartition, the universe set).

This problem is well-known to be NP-hard [Gon85], and we can show the NP-hardness of exactly solving MINETREX by reducing from this problem. It will be, however, more convenient to formulate X3C in a graph-theoretic way for the purpose of this reduction. Observe that we can represent our instance by a bipartite graph of vertex set $A \cup B$ (the two bipartitions), where A represents the sets, and B represents the universe. We add an edge $u \to v$ between the vertex $u \in B$ representing i and the vertex $v \in A$ representing S_j if and only if $i \in S_j$. The goal is now to select n of the vertices in A, so that every vertex in B has exactly one selected neighbour.

We picture our reduction in Figure 4. We hang an arbitrary binary tree on the left part of the graph such that A forms its leaves, where all edges are oriented towards the leaves. To better understand the properties of entropy, we recall the following lemma. Recall $H(d_1, \ldots, d_n) = \sum_{v \in V} d_v \log_2(m/d_v)$.

Lemma 1.9 (Domination, see e.g., [CFJ08, Lemma 1]): *Suppose that $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ are sequences of non-negative real numbers with $x_1 + \cdots + x_n = y_1 + \cdots + y_n$, and let $x_1 \geq \cdots \geq x_n$ and $y_1 \geq \cdots \geq y_n$. We say x* dominates *y if $x_1 + \cdots + x_i \geq y_1 + \cdots + y_i$ for all $i \in [n]$, and x* strictly dominates *y if additionally $x \neq y$.*

If x strictly dominates y, then $H(x_1, \ldots, x_n) < H(y_1, \ldots, y_n)$. ◁

In particular, note that the total number of edges in any subgraph of our instance resulting from the elimination of a spanning tree is the same. Furthermore, note that the maximum indegree of any vertex is 4, and the vertices with indegree 4 have outdegree 0, and thus after eliminating a spanning tree have degree at most 3. Thus the most dominant (in the sense of Lemma 1.9) sequence of indegrees has form $(3, \ldots, 3, 0, \ldots, 0)$. The only way this can happen is if (i) we extract every edge completely within the added tree, and (ii) every vertex in the left hand side has either exactly 1 or exactly 4 edges extracted. Suppose S is the subset of vertices on the left side where we take 4 neighbouring edges. Note that since the subtree we extract is a spanning tree, it must be the case that the root of the tree we added in the reduction is connected to every vertex on the right side. This can only happen via the vertices in S; in particular, by a simple calculation we see that every vertex on the right has exactly one vertex in S as a neighbour.

Thus, we have shown that the optimal possible solution to the reduced MINETREX instance *must* correspond to a solution to the original X3C instance, and vice versa. This implies the hardness of exactly solving MINETREX (and the proof for U-MINETREX is quite similar). Our proofs for Theorem 1.4 and Theorem 1.5 use essentially the same reduction, but starting from a gap version of X3C (actually exact cover with different regularity parameters, not 3). The soundness analysis of our reduction is also more involved, since we must show that we can decode an approximate solution of (U-)MINETREX.

1.4. Open Problems

Since all components of tree extraction are relatively simple, it is conceivable to work well in practice. We reserve a proper empirical investigation of the potential of tree extraction for a future work.

While we focused our attention entirely on trees, the basic idea of tree extraction easily extends beyond trees: extract some subgraph G' from G that can be stored using a constant number of bits per edge and use the (implicit) relabeling for $G - G'$; if we additionally can efficiently find a maximum size subgraph (in terms of number of edges), we may further improve the space savings.

Outline. The remainder of this paper is organised as follows. Section 2 describes our MINETREX approximation algorithm and the generic analysis framework for entropy minimisation. Section 3 gives the inapproximability reductions. Section 4 describes our ultrasuccinct TREX data structure that adds efficient query support to unlabelled graphs; Section 5 extends this construction by first removing twins in a graph. In Section 6, we prove that large classes of graphs indeed allow for significant space savings via tree extraction. In the appendix, we give details of the used NP-hard gap problem from [GT05] (Appendix A), show that there are always $\subseteq$-maximal optimal solutions for MINETREX (Appendix B), and discuss instance-optimal compression for the copy model (Appendix C).

2. Greedy Approximation

In this section we prove Theorem 1.1, as well as Corollary 1.3, thus completing our positive results for (U-)MINETREX. We need the following easy lemma.

Lemma 2.1: *For $d \geq x \geq 0$, we have $(d-x)(\lg d - \lg(d-x)) \leq \frac{x}{\ln 2}$.* ◁

Proof: Recall that $t + 1 \leq e^t$ for all t. There is nothing to prove for $d = x$, so assume $d > x$. Let $t = \ln(d/(d-x)) \geq 0$. We then find that

$$\ln d - \ln(d-x) \quad \leq \quad \frac{d}{d-x} - 1,$$

which, after rearranging and dividing by $\ln 2$, implies our result. □

We restate Theorem 1.1 for the reader's convenience.

Theorem 1.1: *Fix some vector $(d_1, \ldots, d_n) \in \mathbb{N}^n$ with $m = \sum_i d_i$. Fix some positive integer k. Fix also some class*

$$\mathcal{C} \subseteq \left\{ (x_1, \ldots, x_n) \in \mathbb{N} \;\middle|\; \forall i \; 0 \leq x_i \leq d_n \;\wedge\; \sum_i x_i = k \right\}.$$

Consider the following two objective functions:

$$\begin{aligned}\textsc{Opt-Lin}(x_1,\ldots,x_n) &= \sum_i x_i \lg d_i,\\ \textsc{Opt-Ent}(x_1,\ldots,x_n) &= -\sum_i (d_i - x_i)\lg\left(\frac{d_i - x_i}{m-k}\right) = H(d_1 - x_1,\ldots,d_n - x_n).\end{aligned}$$

The following holds for any $(x_1,\ldots,x_n) \in \mathcal{C}$*:*

$$\begin{aligned}&\textsc{Opt-Ent}(x_1,\ldots,x_n) - \frac{k}{\ln 2}\\ &\leq (m-k)\lg(m-k) - \left(\sum_i d_i \lg d_i\right) + \textsc{Opt-Lin}(x_1,\ldots,x_n)\\ &\leq \textsc{Opt-Ent}(x_1,\ldots,x_n).\end{aligned}$$

Hence, if we can minimise Opt-Lin *over* $\mathcal{C}$ *exactly, then we can minimise* Opt-Ent *over* $\mathcal{C}$ *with additive error at most* $k/\ln 2$*.* ◁

Proof: First we apply some standard transformations on the entropy: note that

$$\begin{aligned}\textsc{Opt-Ent}(x_1,\ldots,x_n) &= -\sum_i (d_i - x_i)\lg\left(\frac{d_i - x_i}{m-k}\right)\\ &= \sum_i (d_i - x_i)\log(m-k) - \sum_i (d_i - x_i)\lg(d_i - x_i)\\ &= (m-k)\log(m-k) - \sum_i (d_i - x_i)\lg(d_i - x_i). \qquad (1)\end{aligned}$$

By Lemma 2.1, and by simple monotonicity, for every $i \in [n]$ we find that

$$0 \leq (d_i - x_i)(\lg d_i - \lg(d_i - x_i)) \leq \frac{x_i}{\ln 2}.$$

By summation over all i, and since $\sum_i x_i = k$, we have

$$0 \leq \sum_i (d_i - x_i)(\lg d_i - \lg(d_i - x_i)) \leq \frac{k}{\ln 2},$$

or equivalently

$$-\sum_i (d_i - x_i)\lg d_i \leq -\sum_i (d_i - x_i)\lg(d_i - x_i) \leq \frac{k}{\ln 2} - \sum_i (d_i - x_i)\lg d_i.$$

Finally, we can recognise Opt-Lin in this expression to deduce

$$\begin{aligned}-\left(\sum_i d_i \lg d_i\right) + \textsc{Opt-Lin}(x_1,\ldots,x_n) &\leq -\sum_i (d_i - x_i)\lg(d_i - x_i)\\ &\leq \frac{k}{\ln 2} - \left(\sum_i d_i \lg d_i\right) + \textsc{Opt-Lin}(x_1,\ldots,x_n).\end{aligned}$$

Apply this to the expression found in (1) to find that

$$\begin{aligned}&(m-k)\log(m-k) - \sum_i d_i \lg d_i + \textsc{Opt-Lin}(x_1,\ldots,x_n)\\ &\leq \textsc{Opt-Ent}(x_1,\ldots,x_n)\\ &\leq \frac{k}{\ln 2} + (m-k)\log(m-k) - \sum_i d_i \lg d_i + \textsc{Opt-Lin}(x_1,\ldots,x_n), \qquad (2)\end{aligned}$$

which is exactly the bound we wanted to prove.

Now, let us show that minimising Opt-Lin over $\mathcal{C}$ exactly implies being able to minimise Opt-Ent over $\mathcal{C}$ with additive error at most $k/\ln 2$. For what follows, let

$$\begin{aligned} C &= (m-k)\lg(m-k) - \left(\sum_i d_i \lg d_i\right) \\ x^* &= \arg\min_{x\in\mathcal{C}} \textsc{Opt-Ent}(x) \\ \hat{x} &= \arg\min_{x\in\mathcal{C}} \textsc{Opt-Lin}(x). \end{aligned}$$

It suffices to prove that $\textsc{Opt-Ent}(\hat{x}) \leq \textsc{Opt-Ent}(x^*) + k/\ln 2$. But indeed,

$$\begin{aligned} \textsc{Opt-Ent}(\hat{x}) &\leq \frac{k}{\ln 2} + C + \textsc{Opt-Lin}(\hat{x}) && (2) \\ &\leq \frac{k}{\ln 2} + C + \textsc{Opt-Lin}(x^*) && (\text{Optimality of } \hat{x}) \\ &\leq \frac{k}{\ln 2} + \textsc{Opt-Ent}(x^*) && (2) \end{aligned}$$ □

Next, let us prove that we can solve U-MINETREX with additive error $(m+n-1)/\ln 2$.

Corollary 2.2: *U-MINETREX can be approximated with additive error* $(m+n-1)/\ln 2$ *in* $O(n+m)$ *time.* ◁

Proof: In U-MINETREX, we are given an undirected graph G and must (i) orient every edge, and (ii) delete a spanning subtree so that the indegree entropy of the resulting graph is as small as possible. An equivalent formulation is the following: given a *directed* graph G where the edges come in pairs of form $\{(i,j),(j,i)\}$, (i) delete one edge from every pair, and (ii) delete a spanning subtree from the resulting graph so that the indegree entropy of the resulting graph is as small as possible.

This problem now readily falls within the framework of problems covered by Theorem 1.1. Let $d_1,\ldots,d_n$ be the initial indegrees of the vertices of G. Let $\mathcal{G}$ denote the set of subgraphs of G which consist of (i) exactly one edge from each pair $\{(i,j),(j,i)\}$ in G, together with (ii) a disjoint spanning tree of G. Let $\mathcal{C}$ consist of the indegree sequences of graphs from $\mathcal{G}$. Note that every such graph contains exactly $m+n-1$ edges, and hence the sum of every vector in $\mathcal{C}$ is $m+n-1$. U-MINETREX is now minimising Opt-Ent over $\mathcal{C}$.[8]

Applying Theorem 1.1, we see that in order to solve U-MINETREX with additive error $(m+n-1)/\ln 2$, it is sufficient to find a graph from $\mathcal{G}$ with minimum cost, where the cost of an edge (i,j) is given by $\lg d_j$. Consider a subgraph $H \in \mathcal{G}$ and pair of edges $\{(i,j),(j,i)\}$. In the case that exactly one of (i,j) and (j,i) belongs to H, it is easy to see that the graph otherwise agreeing with H but containing the other edge among (i,j) and (j,i) also belongs to $\mathcal{G}$. Since at least one of these edges must belong to H, we see that in the graph that minimises Opt-Lin, we always take the edge with smaller cost from among these two (and perhaps the other one).

We thus see that it is safe to delete the lower cost edge from the pair $\{(i,j),(j,i)\}$ – that is, the edge pointing towards $\arg\min_{k\in\{i,j\}} d_k$. After this, all that there remains to do is to find a minimum cost spanning tree in the remaining graph. As detailed in the proof of Corollary 1.2, this can be done in $O(n+m)$ time. □

[8] To be precise, we must also find the subgraph $H \in \mathcal{G}$ that gives rise to the minimising indegree sequence, but this is easy to do in our case.

3. Hardness of Approximation

In this section, we show, if $\mathsf{P} \neq \mathsf{NP}$, there exists a constant $\delta > 0$ such that (U-)MINETREX is impossible to approximate in polynomial time up to an additive error of δn bits, where n is the number of vertices in the graph instance. To do so, we aid ourselves with two preliminary problems which we show to be NP-hard; the second will be used as a reduction tool for (U-)MINETREX. We define those problems below.

First, a regular version of gap exact hitting set. Often exact hitting set is formulated as a constraint satisfaction problem: we are given a set of Boolean variables, and a set of constraints on those variables. A constraint is satisfied if and only if exactly one of the variables it is applied to is set to true, and the rest to false ("1-in-k SAT"). We prefer, for linguistic convenience, an equivalent formulation based on bipartite graphs. Transform the variables from the previous formulation into the vertices on the left side of a graph, and the constraints into the vertices on the right side; add edges for all variable-constraint incidences.

We are thus given a bipartite graph $(A \cup B, E)$, and our goal will be to "hit" vertices from B by choosing some vertices from A. Our goal is to hit as many vertices in B as possible; however, a vertex is only "hit" if *exactly one* of its neighbours is in the selected solution (cf. "exact hitting set"). Note that (r, k)-biregularity of the bipartite graph corresponds to each variable in the constraints satisfaction instance being contained within exactly r constraints, and each constraint containing exactly k variables. Note further, that a perfect solution (where all of B is hit) corresponds to an *exact cover* of B; A here is a collection of r-subsets of B. This (arguably more widely known) formulation suffices for the non-approximate hardness (cf. Section 1.3).

Definition 3.1: *In the α-approximate (r, k)-biregular exact hitting set problem (abbreviated (α, r, k)-APX-EHS), we are given an undirected (r, k)-biregular bipartite graph $G = (A \cup B, E)$. A prospective solution to this problem is a subset $S \subseteq A$. We define the value $\xi(S)$ of a solution S to be the fraction of vertices in B which have exactly one neighbour in S. The goal of the problem is to decide whether there exists a solution with value 1, or whether none exists even with value at least $1/\alpha$.* ◁

(α, r, k)-APX-EHS

Input: $G = (A \cup B, E)$, a (r, k)-biregular bipartite graph.

Solution: A subset $S \subseteq A$.

Value: $\xi(S) = \frac{1}{|B|}|\{b \in B : |S \cap N(b)| = 1\}|$.

Yes-inst.: There exists a solution S with $\xi(S) = 1$.

No-inst.: All solutions S have $\xi(S) < 1/\alpha$.

Reverting to the constraint satisfaction view, in this problem we are given an instance, and must decide whether it is perfectly solvable, or we cannot even satisfy a $1/\alpha$ fraction of the constraints in the instance. Guruswami and Trevisan [GT05] prove that a variant of this problem, namely one where the r-regularity of A is omitted, is NP-hard for every $1 \leq \alpha < e$ and for some k depending only on α. A careful reading of their proof shows that their reduction only produces instances with this r-regularity property, where r again depends only on α. Hence we deduce the following.

Theorem 3.2 ([GT05], Theorem 10): *Let e denote the base of the natural logarithm. For any $\alpha < e$, there exist constants r, k such that (α, r, k)-APX-EHS is NP-hard.* ◁

For completeness, we walk through the proof of [GT05] in Appendix A, highlighting the fact that the output instance is r-regular for some r that depends only on α.

Corollary 3.3: *There exist integers $r_2 \geq 10k_2$ such that $(2, r_2, k_2)$-APX-EHS is NP-hard.* ◁

Proof: First, note that integers r, k such that this is true exist by Theorem 3.2; it is not immediate however that $r \geq 10k$. However, fix some arbitrary d, and observe that if $G = (A \cup B, E)$ is an (r, k)-biregular bipartite graph, an input to (α, r, k)-APX-EHS, then the graph $G' = (A \cup ([d] \times B), F)$ where $F = \{(a, (i, b)) \mid (a, b) \in E, i \in [d]\}$ is an instance equivalent to G. This is because the set of solutions is the same, and furthermore (as can be seen by a simple calculation) the value of every solution is the same.

Hence, we see that (α, r, k)-APX-EHS reduces to (α, dr, k)-APX-EHS for any constant d. Taking in particular any $d \geq 10k/r$ completes the proof. □

It turns out that this problem is somewhat inconvenient for our purposes. Considering again the constraint satisfaction view: an approximate solution to the previous problem satisfies a $1/\alpha$-fraction of all the constraints. However, each variable may satisfy or fail to satisfy various constraints. We will prefer a formulation of exact hitting set where we *delete* as few variables as possible, as well as all the constraints incident to them, while attempting to reach an instance which is perfectly solvable. This variant concentrates the "bad" constraints around a small set of "bad" vertices. By analogy to "almost 3-colouring" (see, e.g. [DKPS10]) we name this problem *Almost Exact Hitting Set*. We now define this problem, in the bipartite graph formulation.

Definition 3.4 ((β, r, k)-Almost-EHS): *In the β-almost (r, k)-biregular hitting set problem (abbreviated (β, r, k)-Almost-EHS), we are given a (r, k)-biregular bipartite graph $G = (A \cup B, E)$. A prospective solution to the problem is a pair of sets $S, Z \subseteq A$ such that every $b \in B \setminus N(Z)$ has exactly one neighbour in S. The value of a solution (S, Z), denoted by $\lambda(S, Z)$, is given by $|Z|/|A|$. We must decide whether there exists a solution with value 0, or not even any solution with value at most $1/\beta$.* ◁

(β, r, k)-Almost-EHS

Input: $G = (A \cup B, E)$, a (r, k)-biregular bipartite graph.

Solution: Subsets $S, Z \subseteq A$ satisfying, for all $b \in B \setminus N(Z)$, that $|S \cap N(b)| = 1$.

Value: $\lambda(S, Z) = \dfrac{|Z|}{|A|}$.

Yes-inst.: There exists a solution (S, Z) with $\lambda(S, Z) = 0$.

No-inst.: All solutions (S, Z) have $\lambda(S, Z) > 1/\beta$.

It is not difficult to reduce APX-EHS to Almost-EHS – indeed, this reduction can be generically applied to any constraint satisfaction problem where every variable is contained within the same number of constraints, and every constraint contains the same number of variables.

Theorem 3.5: *Let $\beta_2 = 2k_2$. Then, (β_2, r_2, k_2)-Almost-EHS is NP-hard.* ◁

Proof: We reduce from $(2, r_2, k_2)$-APX-EHS.

Reduction. Our reduction is the "trivial reduction": it does not change the graph, taking as input the (r_2, k_2)-biregular bipartite graph G, and returning G. Obviously this is in polynomial time.

Completeness. Let G be a YES-instance of $(2, r_2, k_2)$-APX-EHS; that is, there exists $S \subseteq A$ with $\xi(S) = 1$. In other words, every vertex in B has exactly one neighbour in S. This implies that $(S, \emptyset)$ is a valid solution to (β_2, r_2, k_2)-ALMOST-EHS. Noting that $\lambda(S, \emptyset) = 0$ implies completeness.

Soundness. Suppose (S, Z) is a solution to (β_2, r_2, k_2)-ALMOST-EHS, with value at most $1/\beta_2$ i.e. $|Z| \leq |A|/\beta_2$. First, observe that by biregularity, $r_2|A| = k_2|B|$, hence we can deduce that

$$|N(Z)| \;\leq\; r_2|Z| \;\leq\; \frac{r_2|A|}{\beta_2} \;=\; \frac{k_2|B|}{2k_2} \;=\; \frac{|B|}{2}.$$

Now, by definition, every $b \in B \setminus N(Z)$ is hit by S i.e. has exactly one neighbour in S. Hence, $\xi(S) \geq \frac{1}{|B|}|B \setminus N(Z)| \geq \frac{1}{2}$. This completes the soundness proof. □

Finally, let us give a gap-version of (U-)MINETREX. The main point here is that this gap version is no harder than approximately solving MINETREX with additive error δn.

δ-MINETREX

Input: $G = (V, E)$, a weakly connected directed graph, and $t \in \mathbb{R}_{\geq 0}$.

Solution: A spanning tree $T \subseteq E$.

Yes-inst.: There exists a solution T with $H(G - T) \leq t$.

No-inst.: All solutions T have $H(G - T) > t + \delta n$.

Theorem 3.6: *δ-MINETREX is NP-hard for some small enough δ. Hence it is NP-hard to approximate MINETREX with additive error δn.* ◁

Proof: We reduce from (β_2, r_2, k_2)-ALMOST-EHS. For brevity, let $\beta = \beta_2, r = r_2, k = k_2$.

Reduction. Let $G_\beta = (A \cup B, E_\beta)$ be a graph instance of (β, r, k)-ALMOST-EHS. Let $n_A = |A|, n_B = |B|$. Construct a digraph G as follows: G starts from G_β, orienting all edges from B to A. Finally, add a binary tree T_A whose leaves are A and otherwise uses new vertices, with edges oriented towards the leaves. Let $n = (2n_A - 1) + n_B$ be the number of vertices in G.

Let $m' = n_B(k - 1)$ and $s(d) = \lg(m'/d)$. It is easy to check that, in any solution T to our δ-MINETREX instance, there are in total m' edges left in $G - T$. Furthermore, if the indegree sequence is given by $d_1 \geq \cdots \geq d_n$, we have $H(G - T) = \sum_i d_i s(d_i)$. Finally, let $t = m's(r)$.

Completeness. Suppose $G_\beta = (A \cup B, E_\beta)$ is a YES-instance of (β, r, k)-ALMOST-EHS; that is there exists $S \subseteq A$ such that every vertex in B has exactly one neighbour in S. Then, G is a YES-instance of δ-MINETREX.

Indeed, construct the spanning tree T as follows: T consists of the edges of the binary tree T_A we added, together with all edges incident to S. T is now a spanning tree, since it connects the root of T_A to all of T_A (and hence A) by construction of T_A, and to all of B, since every vertex in B is adjacent to exactly one vertex in S. It is a tree since (i) no cycle can be formed

within T_A, and (ii) all the vertices in B have (undirected) degree 1 in T, and hence cannot be part of a cycle.

The sorted indegree sequence of $G - T$ is given by $(r, r, \ldots, r, 0, 0, \ldots, 0)$. It is easy to compute, then, that $H(G - T) = m's(r)$, as required.

Soundness. Suppose T is a solution to δ-MINETREX with value at most $t + \delta n$; thus, $H(G - T) \leq m's(r) + \delta n$. As a preliminary step, let us argue that we can always assume that every edge within T_A is included in T. We do this in two steps.

1. Consider any edge (u, v) in T_A where $v \notin A$, and (u, v) *not* in T. If this edge is not included in T, then the indegree of v in $G - T$ is by construction 1. A simple calculation shows that it never increases $H(G - T)$ to add (u, v) to T, and to remove any edge that closes a cycle in T. In particular, since T_A is a tree, it is always possible to have (u, v) replace an edge not belonging to T_A.

2. Next consider an edge (u, v) in T_A where $v \in A$, but (u, v) *not* in T. Observe that all the neighbours[9] of v in T are in B. Hence, since T is a spanning tree, there exists some edge $(w, v) \in T$ with $w \in B$ such that (u, v), (w, v) and a simple path from u to w form a cycle. Adding (u, v) to T and removing (w, v) from T leaves us with a spanning tree which gives rise to the same sequence of indegrees, thus not changing $H(G - T)$.

Hence we see that the edges of T_A can be assumed to belong to T.

Next we claim that for some constant C that depends only on k, r, we have that $G - T$ contains at most $\delta n / C$ vertices v with indegree $0 < d_v < r$.

By construction, no vertex can have indegree $> r$ in $G - T$. Thus, the sorted indegree sequence of $G - T$ must have form $r = \cdots = r > d_i \geq \cdots \geq d_n$. Let ℓ be the number of appearances of r in this sequence; by Lemma 1.9, we have that the minimum-entropy indegree sequence that begins with ℓ copies of r is given by

$$\underbrace{r = \cdots = r}_{\ell} \;>\; \underbrace{r - 1 = \cdots = r - 1}_{\left\lfloor \frac{m' - r\ell}{r-1} \right\rfloor} \;>\; d_0 \;\geq\; 0 = \cdots = 0,$$

where $0 \leq d_0 < r - 1$ is given by the remainder of dividing $m' - r\ell$ by $r - 1$. Recalling that $\left\lfloor \frac{m'-r\ell}{r-1} \right\rfloor = \frac{m'-r\ell-d_0}{r-1}$, we see that the entropy of this sequence is given by

$$\begin{aligned}
& r\ell s(r) \;+\; (m' - r\ell - d_0)s(r-1) \;+\; d_0 s(d_0) \\
&= \; m's(r-1) \;-\; r\ell(s(r-1) - s(r)) \;+\; d_0(s(d_0) - s(r-1)) \\
&= \; m's(r) \;+\; m'(s(r-1) - s(r)) \;-\; r\ell(s(r-1) - s(r)) \;+\; d_0(s(d_0) - s(r-1)) \\
&= \; m's(r) \;+\; (m' - r\ell)(s(r-1) - s(r)) \;+\; d_0(s(d_0) - s(r-1)) \\
&\leq \; m's(r) \;+\; \delta n \qquad\qquad \text{(by assumption).}
\end{aligned}$$

Hence, letting $C = s(r-1) - s(r) = \lg(r/(r-1))$, a positive constant, and $D = d_0(s(d_0) - s(r-1)) = d_0 \lg((r-1)/d_0)$, a non-negative number, we have

$$C(m' - r\ell) + D \;\leq\; \delta n,$$

or equivalently

$$m' - r\ell \;\leq\; \frac{\delta n - D}{C} \;\leq\; \frac{\delta n}{C}.$$

[9] We say that u and v are neighbours in some *directed* graph if and only if at least one of (u, v) or (v, u) are edges in that graph.

Now, since $G - T$ has m' edges left, and $r\ell$ of those are pointing towards vertices of indegree r, this means that at most $m' - r\ell \leq \delta n/C$ of them are pointing towards vertices of indegree less than r. In particular, if each of these points towards a different vertex, we see that there are at most $\delta n/C$ vertices in $G - T$ with indegree between 1 and $r - 1$, proving the claim we began with.

Next, observe that $rn_A = kn_B$, and furthermore $n = (2n_A - 1) + n_B$. Hence $n = (2 + (r/k))n_A - 1 \leq (2 + (r/k))n_A$, and furthermore there are at most $\delta(2 + (r/k))n_A/C$ vertices in $G - T$ with indegree between 1 and $r - 1$. Set δ small enough so that this number is at most n_A/β.

We now claim that setting S to be the set of vertices in A with indegree 0 in $G - T$, and Z to be the set of vertices in A with indegree between 1 and $r - 1$ in $G - T$ is a valid solution to the original (β, r, k)-Almost-EHS instance, with value $\lambda(S, Z) = |Z|/n_A \leq 1/\beta$. To see why this is the case, consider any vertex $u \in B$ that does not have a neighbour in Z. Considering only the adjacencies in T now, we see that u must have a non-leaf neighbour $v \in A$.[10] Note then that v has indegree at least 2 in T; hence in $G - T$ it has indegree at most $r - 1$. But by assumption $v \notin Z$; hence the indegree of v in $G - T$ is 0, and $v \in S$ – so we have shown that every vertex in $B \setminus N(Z)$ has *at least* one vertex in A as a neighbour.

To see why any vertex $u \in B \setminus N(Z)$ has *at most* one neighbour in S, suppose u has two such neighbours $v, w \in S$. Then this induces a cycle in T, since v and w are connected via T_A (as all of T_A is included in T). This is not possible since T is a tree. □

The undirected case is at least equally as hard as the directed one. Our reduction is identical, as is the completeness analysis. The soundness analysis is somewhat more difficult; we sketch the necessary changes:

- It is a bit more difficult to insure that T_A is included in T. In particular, our argument relied on the fact that vertices from T_A but not in A can only have indegree 0 or 1 in $G - T$. Now the vertices internal to the tree T_A may have indegree 2. Nevertheless, there are at most $\Theta(\delta n)$ vertices with indegree 1 or 2 in $G - T$ (as $r \geq 10k > 2$). Consider any such vertex u, and suppose that it has two edge pointing towards it in $G - T$, namely (u, v) and (w, v). If we were to add (u, v) and (w, v) to T, and remove two edges not from T_A to eliminate the created cycles, this can only increase $H(G - T)$ by $\Theta(1)$. This is because the difference in cost for the edges we add and remove themselves is at most $\lg(r) = O(1)$ for each addition/removal pair. The edges we remove from T may also affect the costs of other edges – but since this only increases indegrees in $G - T$, it only *lowers* their costs. Hence overall, we may assume that T_A is included in T with at most a penalty of $\Theta(\delta n)$ – thus the rest of our reduction works, so long as we set δ even smaller.

- Since $r \geq 10k$, we see again that the maximum indegree that could be achieved in the graph, after removing a spanning tree, is r. In the directed case B had indegree 0 no matter what we did; here it is necessary to observe that the indegrees of B are never $\geq r$, no matter how we orient the graph.

- Finally, observe that the graph we used in our reduction has constant maximum (undirected) degree. Hence, n and m are asymptotically equivalent, and we may freely go from a linear error in n to a linear error in m.

Hence we deduce the following.

[10] This is because u must be connected to the root of T_A through the spanning tree T.

Corollary 3.7: *δ-U-MINETREX is NP-hard for all small enough δ – even on graphs with constant maximum (undirected) degree. Hence it is NP-hard to approximate U-MINETREX with additive error δn, or with additive error $\delta' m$ for some constants δ, δ'.* ◁

4. Ultrasuccinct Unlabelled Graphs

In this section, we show how to support efficient navigational queries on our TREX-compressed graph representation, turning TREX into a *data structure* for unlabelled graphs. We extend the techniques from [INW25] from preferential-attachment graphs to *general unlabelled graphs*, both undirected and directed.

4.1. Preliminaries

For the reader's convenience, we recall the results on succinct data structures that we build on in this section. First, we cite the compressed bit vectors of Pătrașcu [Păt08]. (A more practical variant with worse space exists, as well [RRRS07].)

Lemma 4.1 (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 $S \in [\sigma]^n$ while compressing them to zeroth-order empirical entropy

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

Here, $|S|_c$ is the number of occurrences of character c in S.

Lemma 4.2 (Wavelet tree [Nav14]): *Let $S[1..n]$ be an 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.* ◁

We lastly need a succinct data structure for ordinal trees with support for levelorder indices (BFS visit order). Since we only need very basic queries, the classical LOUDS representation [Jac89] is sufficient. (Further operations can be supported via tree covering [HMN+20].)

Lemma 4.3 (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 levelorder 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;*
- *child-rank*(T, v, c): *assuming c is a child of v, return the index i so that $c = \mathsf{child}(T, v, i)$.* ◁

4.2. Adjacency string representation

We begin with a space-efficient variant of the textbook adjacency-list representation for *labelled* graphs, which we will build on below.

Given a directed graph $G = (V, E)$ with $V = [n]$, we define the *adjacency string* $A = A(G)$ as the concatenation of the outneighbourhoods of all vertices. Formally, $A(G) = [N^+(v_1), N^+(v_2), \ldots, N^+(v_n)]$, where $N^+(v) = \{u \in V : (v, u) \in E\}$ denotes the outneighbourhood of vertex v; we similarly define $N^-(v) = \{u \in V : (u, v) \in E\}$. Note that A crucially depends on the order/names of vertices.

Note further that A itself does not allow to reconstruct G since it does not encode the boundaries between neighbourhoods resp. the outdegrees d_v^{out} of vertices. We thus augment A with a bitvector $S[1..n+m] = S(G)$ encoding the unary degree sequence $d_1^{\text{out}}, \ldots, d_n^{\text{out}}$ (or equivalently, the starting indices of the next neighbourhood) as follows: for $v = 1, \ldots, n$ append to S one 1 followed by d_v^{out} many 0s. So S contains exactly n one-bits.

Using compressed bitvectors (see Lemma 4.1), S can be stored with $n \lg(m/n) + O(n)$ bits of space, which is never worse than the $\Theta(n \log n)$ bits of space used by the n pointers in a standard adjacency-list representation, and substantially better if G is sparse. Moreover, we can find the starting index of any outneighborhood using rank and select (details below).

4.3. TREX Data Structure

We will now describe our data structure for compressing unlabelled directed graphs.

Let $G = (V, E)$ be a directed graph on $n = |V|$ vertices and $m = |E|$ edges. As a first compression step, we compute the tree extraction spanning tree T in G using our MST-approximation (Section 2). The vertices of the spanning tree T are the same as G, i.e., $V(T) = V$. We will store T using Lemma 4.3. For this, we can root T arbitrarily and choose an ordering of children arbitrarily. Nodes in T are referred to by their levelorder indices. Note that in this ordering all children of a node in T form an *interval* in the levelorder indices. We do a level order traversal of the spanning tree T and compute the renaming mapping for vertices of G according to their level order indices from $\{1, \ldots, n\}$.

We now by represent $G - T$ using the adjacency-string encoding from above (Section 4.2), using the levelorder indices from T to order the vertices. Let $A' = A(G - T)$ and $S' = S(G - T)$ denote the adjacency string and unary outdegree sequence of $G - T$, respectively. Compared to $A(G)$, the sequence A' contains exactly $n - 1$ fewer vertex symbols (and similarly for S'). The residual sequence A' is stored using a wavelet tree (Lemma 4.2) over the alphabet $\{1, 2, \ldots, n\}$.

Edge orientations are implicitly stored in our representation, as adjacency lists store only outgoing edges. However, we have to explicitly store the direction of tree edges since T is an undirected spanning tree. We thus store a bitvector D, indexed so that $D[v]$ is 1 iff v's parent edge is pointing towards (i.e. down to) v. In Figure 2 (page 4), the ultrasuccinct TREX data structure on a directed graph with 8 vertices is illustrated.

For undirected graphs, we find the spanning tree *and orientation of the edges of $G - T$*, but we do not need to store the direction of tree edges. Otherwise we follow the same procedure.

4.4. Operations

Recall that we store $G = (V, E)$, where $|V(G)| = n$ and $E(G) = m$, using an ordinal tree T, a wavelet tree $A' = A(G - T)$ and a bitstring $S' = S(G - T)$ of length $m - (n - 1) + n = m + 1$ with n many 1s to mark the start of the adjacency list of each vertex in A'.

Outdegree of v. The outdegree is the sum of the outgoing edges stored in A' and the outgoing edges stored in T. Note that since T is an unordered spanning tree, any subset of v's child edges and its parent edge may be outgoing edges.

The number of v's outgoing edges in A' is given by $\mathsf{select}_1(S', v + 1) - \mathsf{select}_1(S', v) - 1$. (Here, we assume that $\mathsf{select}_1(S', n + 1) = |S'| + 1 = m + 2$.)

The number of outgoing edges in T is given by the sum of (a) 1 if $D[v] = 0$ (and $v \neq 1$) and (b) the number of 0s in $D[c_1..c_k]$ (computable via $\mathsf{rank}_0(D, c_k) - \mathsf{rank}_0(D, c_1 - 1)$) where $c_1 = \mathsf{child}(T, v, 1)$ and $c_k = \mathsf{child}(T, v, \mathsf{degree}(T, v))$.

Indegree of v. As for the outdegree, we obtain the indegree of a vertex v as the sum of the incoming edges in A' and the incoming edges in T. The latter is entirely symmetric to outdegree, just counting 1s in D instead of 0s.

The indegree in A' equals the number of occurrences of "letter" v in the string A', which is given by $\mathsf{rank}_v(A', m')$ for $m' = |A'| = m - n + 1$.

Deciding if u and v are adjacent. Vertices u and v are adjacent, if and only if either (1) u is the parent of v in T, $\mathsf{parent}(T, v) = u$, (2) vice versa, $\mathsf{parent}(T, u) = v$, (3) v occurs in u's adjacency list in A' or (4) u occurs in v's adjacency list in A'.

Conditions (1) and (2) are directly supported on T; for (3) and (4), we can use rank and select on S' and A' as follows. For condition (3), we first find the index of u in S' using $s = \mathsf{select}_1(S', u) - u$. This marks the start of the adjacency list of u (we have to subtract the u one-bits in S'). Therefore, if v occurs in $A'[s+1 \ldots s+\mathsf{outdegree}(u)-1]$, this implies u and v are adjacent. In other words, if $\mathsf{rank}_v(A', s + \mathsf{outdegree}(u) - 1) - \mathsf{rank}_1(A', s) \geq 1$, then v occurs and u and v are adjacent. Condition (4) is similar.

Computing the ith outneighbour of v ($\mathsf{N_{out}}(v, i)$). Since adjacency lists in general do not follow any specific order, we can define a convenient ordering of neighbours. By convention, we will consider all outneighbours of v in T to precede all outneighbours of v in A'.

The ith outneighbour of v in G can then be determined as follows: If $i \leq d_T$ for d_T the outdegree of v in T, then we find the neighbour in T; otherwise, it is stored in the wavelet tree A'. d_T is computed as in outdegree.

Assume first that $i \leq d_T$. If $i = 1$ and $D[v] = 0$, return the parent of v in T. Otherwise, we have to find the jth outgoing child in T where $j = i$ if $D[v] = 1$ and $j = i - 1$ otherwise. This child is given by $\mathsf{select}_0(D, j + o)$ for $o = \mathsf{rank}_0(D, c_1 - 1)$ with $c_1 = \mathsf{child}(T, v, 1)$.

Assume now that $d_T < i \leq d_v^{\text{out}}$. We have to find the jth outgoing edge from v in A' for $j = i - d_T$. v's outneighbourhood in A' begins at $s = \mathsf{select}_1(S', v) - v$, so we find our neighbour at $\mathsf{access}(A', s + j)$.

Computing the ith inneighbour of v ($\mathsf{N_{in}}(v, i)$). As for outneighbours, we consider inneighbours in T to precede those in A'. Let (with slight abuse of notation) here d_T denote the *in*degree of v in T. Then the case $i \leq d_T$ is symmetric to ith outneighbour.

It remains to consider $d_T < i \leq d_v^{\text{in}}$ and select the jth inneighbour of v in A' for $j = i - d_T$. The jth occurrence of v in A' is found at index $y = \mathsf{select}_v(A', j)$; that index y belongs to vertex $w = \mathsf{rank}_1(S', y)$, so the jth incoming edge originates from w.

Outneighbour rank ($\mathsf{N_{out}}$-rank(v, w)). In preparation of the extension with twin removal (Section 5), we implement the following operation, which may also be of independent interest: $\mathsf{N_{out}}\text{-}\mathsf{rank}(v, w)$, which returns the index i so that $w = \mathsf{N_{out}}(v, i)$, assuming that w is an outneighbour of v in G.

We check similar cases as when finding the ith outneighbour. First, we check if w is v's parent in T and $D[v] = 0$; if so, return 1. If conversely v is w's parent and $D[w] = 1$, we find the number j of out-children of v in T before w via $j = \mathsf{rank}_0(D, w) - \mathsf{rank}_0(D, c_1 - 1)$ with $c_1 = \mathsf{child}(T, v, 1)$. The result is then $j - [D[v] = 0]$. Otherwise w occurs in v's section of A'. To find its position, we compute $j = \mathsf{select}_w(A', r + 1) - s$ where $r = \mathsf{rank}_w(A', s - 1)$ and $s = \mathsf{select}_1(S', v) - v$; then the answer is $j + d_T$ for d_T the outdegree of v in T.

Inneighbour rank ($\mathsf{N_{in}}$-rank(v, w)). Similar to $\mathsf{N_{out}}\text{-}\mathsf{rank}(v, w)$, we also implement inneighbour rank. The checks in T are entirely symmetric to the above, so we only need to consider the case that the edge (w, v) is represented in A', i.e., we are looking for the occurrence of v in w's section, and need to determine the number j of incoming edges, i.e., occurrences of v, up until this occurrence. We compute $j = \mathsf{rank}_v(A', s - 1) + 1$ for $s = \mathsf{select}_1(S', w) - w$ the beginning of w's segment; the answer then is $j + d_T$ where d_T denotes the indegree of v in T (computed as above).

4.5. Space and time analysis

In this section, we prove the space and time complexity of our succinct graph representation, restated here for convenience.

Theorem 1.8 (Ultrasuccinct TREX): *Given a directed graph G and spanning tree T in G, we can construct a data structure for a graph isomorphic to G (where the vertex relabeling can be output at construction time) that occupies $H(G - T) + 3n + o(m) + \lg \binom{m}{n} \leq H(G - T) + n \lg(m/n) + 4.4427n + o(m)$ bits of space, while supporting the operations $\mathsf{N_{out}}(v, i)$ (ith outneighbour of v), $\mathsf{N_{in}}(v, i)$ (ith inneighbour of v), $\mathsf{indegree}(v)$, and $\mathsf{adjacent}(u, v)$ in $O(\lg n)$ time; $\mathsf{outdegree}(v)$ takes $O(1)$ time.* ◁

Proof: The space for A' is $H_0(A') + o(m) = H(G - T) + o(m)$. The space for S' is $\lg \binom{m+1}{n} + O(m/\log^c m) = \lg \binom{m+1}{n} + o(m)$. We have $\lg \binom{m+1}{n} \leq \lg \binom{m}{n} + O(\lg m) \leq n \lg(m/n) + n/\ln 2 \leq n \lg(m/n) + n/\ln 2 + O(\lg m)$. The data structure for T uses $2n + o(n)$ bits of space, and $D[1..n]$ contributes another $n + o(n)$ bits for an uncompressed bitvector with rank and select support.

The running time of operations is dominated by the operations on A': access, rank, and select take $O(\lg \sigma)$ time on the wavelet tree, where our σ is n here. All operations on S', D and T run in constant time. The outdegree of a vertex does not need operations on A'. □

5. Twin Removal

We can further compress certain graphs by first contracting vertices with identical neighbourhood ("twins"). This is particularly effective on unlabelled graphs, since it basically suffices to remember how many twin copies each vertex has. The resulting reduced (twin-free) graph is then represented using Theorem 1.8

We formally define the notion of *twins* as follows: Let $G = (V, E)$ be a graph, where $|V| = N$. For a vertex $v \in V$, let $N(v) = \{u \in V \setminus \{v\} : \{u, v\} \in E\}$ denote the open neighbourhood of v. Two distinct vertices $u, v \in V$ are called *twins* if $N(u) = N(v)$; for directed graphs, two vertices $u, v \in V$ are *twins* if $N^+(u) = N^+(v)$ and $N^-(u) = N^-(v)$. We focus again on digraphs.

Being twins defines an equivalence relation on the vertex set V. Let $V = C_1 \dot\cup C_2 \dot\cup \cdots \dot\cup C_n$ be the resulting partition into *twin classes*, where vertices within the same class are pairwise twins. (This is also formally stated as Lemma 1 of [TMS20]). Contracting all twin classes into single vertices yields the *twin-reduced graph*. More formally, we define $G_0 = (V_0, E_0)$ as follows: The vertex set is $V_0 = \{v_1, \ldots, v_n\}$, where each v_i corresponds to the twin class C_i. We have an edge $(v_i, v_j) \in E_0$ if and only if $(u, w) \in E$ for some (equivalently, all) $u \in C_i$, $w \in C_j$. By construction, the graph G_0 is *twin-free*, i.e. no two vertices in G_0 have identical open neighbourhoods.

5.1. Operations with twin removal

As we will show, we can implement operations on the original graph G using operations on G_0 as a black box. We only need to store the sizes of the twin classes on top. We use the unary encoding of $[|C_1|, |C_2|, \ldots, |C_n|]$ in a compressed bitvector for that; more precisely, for each twin class C_i, we write a 1-bit in B followed by $|C_i - 1|$ many 0-bits.

While some operations can be supported equally efficiently, an exception is the random access to neighbourhood, i.e., operations $\mathsf{N}_{\mathsf{out/in}}(v, i)$, which the twin-reduced representation can not support efficiently. (We would need prefix sum of (out/in)degrees of vertices in G_0, not just access to their individual degrees.) We can however support a slightly weaker operation that allows equally efficient traversals through the neighbourhood: $\mathsf{N}_{\mathsf{out/in}}\text{-}\mathsf{next}(v, w)$ yields the next out/in neighbour of v *after* w resp. the first neighbour of v if $w = \texttt{null}$. Note that this is indeed what a linked list of outneighbours in a textbook adjacency-list representation offers.

Vertex ids. For a vertex $v \in V(G)$, the vertex in G_0 which is its twin, is called its *representative* $\mathsf{rep}(v)$. For $v \in V(G_0)$, the representative of v is v itself. We represent G_0 using our TREX data structure (Theorem 1.8). Thus the tree T induces a level-order indexing of the vertices of the twin-reduced graph G_0, assigning them the labels $\{1, \ldots, n\}$. Twins are assigned consecutive labels from the range $\{n + 1, \ldots, N\}$, in order of their representatives. This is conveniently done via bitvector B, where each twin corresponds to a 0-bit. For example, if $N = 10$ and $n = 3$, the bitstring $B = \texttt{1000100100}$ indicates that after the three representatives 1, 2, 3, we assign 4, 5, 6 as ids for the 3 twins of 1; ids 7 and 8 for the two twins of 2, and finally 9 and 10 to the two twins of 3.

Representative. For vertex $v \notin V(G_0)$, we obtain its *representative* as

$$\mathsf{rep}(v) \;=\; \mathsf{rank}_1(B, (\mathsf{select}_0(B, v - n))).$$

In other words, we take the number of 0s in B from positions 1 to v, given by $v - n$, and find the index of the $(v - n)$th 0 in B using select, and then count the number of 1s up to that index using rank.

Adjacency queries. From the definition, twins have the same neighbourhood. Therefore, for any two vertices $v, w \in V(G)$, $(v, w) \in E(G)$ if and only if $(\mathsf{rep}(v), \mathsf{rep}(w)) \in E(G_0)$. Thus, adjacency queries between two vertices on G directly translate to adjacency queries on their representatives in the twin-reduced graph G_0.

Twin queries. Given a vertex $v \in V(G_0)$, we can find the labels of its twins using a select query on B. We first find the index of vth $\mathtt{1}$ in B using $i = \mathsf{select}_1(B, v)$. The number of $\mathtt{0}$s before that index is given by $i - v$. This implies that the twins of v are precisely the vertex ids $[n + i - v + 1 \,..\, n + i - v + |C_v| - 1]$.

Next outneighbour of v after w. We iterate through the outneighbours of v by twin classes, i.e., in the neighbourhood order of G_0, but stepping through all twins of a vertex directly after its representative is output.

If w is *not* a last twin copy of its representative $w_0 = \mathsf{rep}(w)$, we simply increment w resp. find the first twin of w_0 in case $w = w_0$. This can be done as in the twin query above.

If w is the last copy of w_0, we now need to advance in G_0 to the next neighbour u_0 of $v_0 = \mathsf{rep}(v)$ after w_0; we find this next neighbour as $u_0 = \mathsf{N_{out}}(v_0, r + 1)$ for $r = \mathsf{N_{out}\text{-}rank}(v_0, w_0)$. (Both operations are in G_0).

Next inneighbour of v after w. Using an entirely symmetric procedure as for outneighbours works; only the queries in G_0 need to use $\mathsf{N_{in}}$ instead of $\mathsf{N_{out}}$.

***i*th (out/in)neighbour.** Unless G_0 is much smaller than G, i.e., G has very many twins, using this random access operation on neighbourhoods is very slow and should be avoided. For completeness, we sketch how to support it, nevertheless.

Assuming that, for $i' \leq n - 1$, we know how to query the i'th outgoing and i'th incoming vertices in G_0, then, for $i \leq N - 1$, we can query the ith outgoing and ith incoming vertices in G by iterating over the neighbours in G_0 until we reach the correct range for i. Note that each neighbour in G_0 can correspond to an arbitrary number of twin neighbours which need to be counted for i. The operation is entirely symmetric between out- and inneighbours, so we generically speak of neighbours here.

For a vertex $v \in V(G)$, let $u \in V(G_0)$ be its representative. We query for the first neighbour of u in G_0, namely x_1. If $i > |C_{x_1}|$, none of the twins of x_1 will be the ith neighbour of v. Let $y_1 = i - |C_{x_1}|$. We again query for the second neighbour of u, namely x_2. If $y_1 > |C_{x_2}|$, none of the twins of x_2 will be the ith neighbour of v. We continue this process until, for some neighbour x_j of u, we get $|C_{x_j}| > y_{j'}$. The $y_{j'}$th twin of x_j will be the ith neighbour of the vertex v. We can use the twin query to get the label of the ith vertex.

Degree queries. In a similar spirit to the ith out/in neighbour query as above, the degree query for a vertex v is not efficiently supported. This is because we need to sum the number of twins over the neighbourhood of $\mathsf{rep}(v)$ in G_0. We sketch the details for completeness.

Let $v \in V(G)$ and let $i = \mathsf{rep}(v)$. The outdegree and indegree of v in G can be expressed as $|N^+(v)| = \sum_{(i,j)\in E(G_0)} |C_j|$ and $|N^-(v)| = \sum_{(j,i)\in E(G_0)} |C_j|$, respectively. In particular, computing degrees in G reduces to counting the neighbours of the corresponding representative in G_0, weighted by the sizes of the associated twin classes.

Specifically, for the outdegree of a vertex v, find all the outneighbours of $i = \mathsf{rep}(v)$ in G_0; in other words, if $N^+(i) = \{\, j \mid (i,j) \in E(G_0) \,\}$, then for each $j \in N^+(i)$, the number of vertices belonging to the jth class is determined using select queries on the bitvector B as follows:

$$|C_j| \;=\; \begin{cases} \mathsf{select}_1(B, j+1) - \mathsf{select}_1(B, j) - 1 & j < n, \\ |B| - \mathsf{select}_1(B, j) & j = n. \end{cases}$$

In a similar way, we can calculate the indegree and total degree of a vertex in G.

5.2. Space and time analysis

In this section, we prove the space and time complexity of our twin-removal data structure.

Theorem 5.1 (Twin removed TREX): *For a directed graph G, there exists a data structure for the twin-removed TREX representation – consisting of B, T, A' and S – using $H_0(A') + 3n + o(m+N) + \lg\binom{m}{n} + \lg\binom{N}{n}$ bits of space, while allowing for operations* $\mathsf{adjacency}(u,v)$ *to run in $O(\lg n)$ time and iterating through* $\mathsf{N_{out}}(v)$ *and* $\mathsf{N_{in}}(v)$*, in $O(\log n)$ time per neighbour. The operations* $\mathsf{N_{out}}(v,i), \mathsf{N_{in}}(v,i)$*, and* $\mathsf{degree}(v)$ *run in $O(d \lg n) = O(n \lg n)$ time, where d is the number of neighbors of* $\mathsf{rep}(v)$ *in G_0.* ◁

Proof: From Theorem 1.8, we already know the space required by T, A' and S. Recall that bitvector B encodes the multiplicities of twins in G. Using compressed bitvectors (Lemma 4.1), B with n many 1s and $|B| = N$ can be stored in $\lg\binom{N}{n} + o(N) \le n \lg \frac{N}{n} + O\Big(\frac{N}{\lg^c N}\Big)$ bits, for any constant $c > 0$, while supporting rank and select in $O(1)$ time. This proves the space analysis.

The query times for $\mathsf{adjacency}(u,v)$, $\mathsf{N_{out}\text{-}next}(v,w)$ and $\mathsf{N_{in}\text{-}next}(v,w)$ directly follow from Theorem 1.8. On the other hand, for operations $\mathsf{N_{out}}(v,i), \mathsf{N_{in}}(v,i)$, and $\mathsf{degree}(v)$, we have to iterate sequentially over the neighbourhoods in G_0, with a $O(\lg n)$ time query for each of the d representative neighbours. Clearly, $d \le n$. □

6. Good graphs classes

In this chapter, we prove that removing trees often leads to a significant decrease in the number of bits needed to store a graph. First, it will be useful to use a variant of Theorem 1.1 that is tailored for our purposes in this chapter. For the following we will often think of equations holding only up to some error term. For this purpose, when we write for example $x = y \pm \varepsilon$, we mean that $|x - y| \le \varepsilon$.

Theorem 6.1: *Consider the same setup as in Theorem 1.1. Then*

$$H(d_1 - x_1, \ldots, d_n - x_n) \;=\; H(d_1, \ldots, d_n) \;+\; \sum_i x_i \lg \frac{d_i}{m} \;\pm\; \frac{2k}{\ln 2}.$$

◁

Proof: Indeed, noting that $\textsc{Opt-Ent}(x_1, \ldots, x_n) = H(d_1 - x_1, \ldots, d_n - x_n)$, we have, by the main inequality of Theorem 1.1,

$$\begin{aligned} H(d_1 - x_1, \ldots, d_n - x_n) &= \textsc{Opt-Ent}(x_1, \ldots, x_n) \\ &= (m-k)\lg(m-k) - \sum_i d_i \lg d_i + \textsc{Opt-Lin}(x_1, \ldots, x_n) \;\pm\; \frac{k}{\ln 2} \\ &= (m-k)\lg(m-k) - \sum_i d_i \lg d_i + \sum_i x_i \lg d_i \;\pm\; \frac{k}{\ln 2} \end{aligned}$$

Now, apply Lemma 2.1 to $(m-k)(\lg m - \lg(m-k))$ to find that $(m-k)\lg(m-k) =$

$(m-k)\lg m \pm \frac{k}{\ln 2}$, hence

$$\begin{aligned} H(d_1 - x_1, \ldots, d_n - x_n) &= (m-k)\lg m - \sum_i d_i \lg d_i + \sum_i x_i \lg d_i \pm \frac{2k}{\ln 2} \\ &= \sum_i (d_i - x_i)\lg m - \sum_i d_i \lg d_i + \sum_i x_i \lg d_i \pm \frac{2k}{\ln 2} \\ &= \left(-\sum_i d_i \lg \frac{d_i}{m}\right) + \left(\sum_i x_i \lg \frac{d_i}{m}\right) \pm \frac{2k}{\ln 2} \\ &= H(d_1, \ldots, d_n) + \sum_i x_i \lg \frac{d_i}{m} \pm \frac{2k}{\ln 2}. \end{aligned}$$

□

Let us now apply this theorem to our particular case.

Corollary 6.2: *Consider any digraph $G = (V, E)$ with n vertices and m edges. Consider any spanning tree T of G. Assign edge (i, j) weight $w(i, j) = -\lg d_j/m$, where d_j is the indegree of j in G. We have*

$$H(G - T) = H(G) - \sum_{(i,j)\in T} w(i,j) \pm \frac{2n}{\ln 2}.$$

Hence the optimal savings we can gain (up to an error of $\frac{2n}{\ln 2}$) by removing some tree is given by the maximum cost spanning tree under the weights described above. ◁

This is both stronger and weaker than Theorem 1.1 – whereas that theorem told us that linear optimisation is sufficient for approximate entropy minimisation, this theorem also relates the precise value of the entropy associated with deleting a tree with a linear function of the edges remaining in the graph (but with a larger error).

Recall the classic result of [Edm71]: the 0–1 vectors corresponding to spanning trees of a given graph G form the vertices of a polytope with a simple formulation as an LP. More precisely, we can find a maximum cost spanning tree in a graph $G = (V, E)$ with indegree sequence $d_1, \ldots, d_n$, with weights as above, by solving the following linear program:

$$\begin{aligned} \underset{x_e,\, e \in E}{\text{maximize}} \quad & -\sum_{(i,j)\in E} x_{ij} \lg \frac{d_j}{m} \\ \text{subject to} \quad & \sum_{e\in E} x_e = n - 1, \\ & \sum_{e\in E\cap S^2} x_e \le |S| - 1 \quad \forall S \subseteq V, 1 < |S| < n, \\ & x_e \ge 0 \qquad \forall e \in E \end{aligned}$$

It is easy to see that eliminating the first equality constraint leads to a polytope whose vertices are precisely the acyclic subgraphs of G. Since we have only non-negative costs and will only consider connected graphs, we may as well weaken that constraint. This leads to the following equivalent formulation:

$$\begin{aligned} \underset{x_e,\, e \in E}{\text{maximize}} \quad & -\sum_{(i,j)\in E} x_{ij} \lg \frac{d_j}{m} \\ \text{subject to} \quad & \sum_{e\in E\cap S^2} x_e \le |S| - 1 \quad \forall S \subseteq V, |S| \ge 2, \\ & x_e \ge 0 \qquad \forall e \in E \end{aligned} \tag{3}$$

(This can also be seen as an equivalent formulation for the independence polytope of the graphical matroid.) We hence get the following fact, by combining this LP formulation for maximum cost spanning tree, together with Corollary 6.2.

Corollary 6.3: *Consider some connected digraph $G = (V, E)$ with n vertices and m edges, with indegreee sequence $d_1, \ldots, d_n$. Suppose there exists some values $x_e \in \mathbb{R}_{\geq 0}$ for $e \in E$, such that for all $S \subseteq V$ with $|S| \geq 2$ we have*

$$\sum_{e \in E \cap S^2} x_e \quad \leq \quad |S| - 1.$$

Then there exists some spanning tree T such that

$$H(G - T) \quad \leq \quad H(G) + \left(\sum_{(i,j) \in E} x_{ij} \lg \frac{d_j}{m} \right) + \frac{2n}{\ln 2}.$$

◁

This is precisely what we will use to prove the combinatorial facts we want. Recall that $\delta(G)$ is the ratio of the number of edges of G divided by the number of vertices of G, $\delta(G) = |E(G)|/|V(G)|$.

Theorem 1.6: *Consider some connected n-vertex m-edge graph $G = (V, E)$. Let α be some constant so that for any subgraph G' of G, we have $\delta(G') \leq \alpha \delta(G)$. Then, there exists some spanning tree T of G such that*

$$H(G - T) \quad \leq \quad H(G) - \frac{n}{2\alpha} \mathcal{H}^{\text{in}}_{\text{deg}}(G) + \frac{2n}{\ln 2}.$$

◁

Proof: Consider the values $x_e = n/2m\alpha \geq 0$. Let us check it follows the conditions laid out in Corollary 6.3. Consider any subset $S \subseteq V$ with $|S| \geq 2$. Suppose S has n' vertices within it, and m' edges within it. By assumption, $m'/n' \leq \alpha m/n$. So

$$\sum_{e \in E \cap S^2} x_e \quad = \quad \sum_{e \in E \cap S^2} \frac{n}{2m\alpha} \quad = \quad \frac{m'}{2} \left(\frac{\alpha m}{n} \right)^{-1} \quad \leq \quad \frac{m'}{2} \frac{n'}{m'} \quad = \quad \frac{n'}{2} \quad \leq \quad n' - 1,$$

as required. Now let us ensure that the savings is sufficient. Note that (excepting the error of $2n/\ln 2$), we save

$$\begin{aligned} - \sum_{(i,j) \in E} x_{ij} \lg \frac{d_j}{m} &= - \sum_{(i,j) \in E} \frac{n}{2m\alpha} \lg \frac{d_j}{m} \\ &= - \sum_{j \in V} d_j \frac{n}{2m\alpha} \lg \frac{d_j}{m} \\ &= \frac{n}{2\alpha} \left(- \sum_{j \in V} \frac{d_j}{m} \lg \frac{d_j}{m} \right) \\ &= \frac{n}{2\alpha} \mathcal{H}^{\text{in}}_{\text{deg}}(G), \end{aligned}$$

as required. □

Finally, by similar techniques we can show that graphs where many vertices have nonzero indegree must have a high savings.

Theorem 1.7: *Consider some connected n-vertex m-edge graph $G = (V, E)$, where at least n/α vertices have nonzero indegree. Then there exists some spanning tree T of G such that*

$$H(G-T) \;\leq\; H(G) - \frac{n}{2\alpha}\mathcal{H}^{\text{in}}_{\text{deg}}(G) + \frac{2n}{\ln 2}. \qquad \triangleleft$$

Proof: Let $d_1, \ldots, d_n$ be the indegree sequence. Suppose without loss of generality that $d_1 \geq \cdots \geq d_n$. Suppose that $d_1, \ldots, d_k > 0$ and $d_{k+1} = \cdots = d_n = 0$. Hence, by assumption $k \geq n/\alpha$. Consider the sequence $x_{ij} = 1/2d_j$. (Note that for every edge $(i,j) \in E$, $d_j \geq 1$ so this division is well-defined.) As before we check that this satisfies the conditions of Corollary 6.3. Consider any subset $S \subseteq V$ with at least 2 vertices. Note:

$$\sum_{e \in E \cap S^2} x_e \;\leq\; \sum_{\substack{j \in S \\ d_j > 0}} \sum_{(i,j)\in E} x_{ij} \;=\; \sum_{\substack{j \in S \\ d_j > 0}} \sum_{(i,j)\in E} \frac{1}{2d_j} \;=\; \sum_{\substack{j \in S \\ d_j > 0}} d_j \frac{1}{2d_j} \;\leq\; \frac{|S|}{2} \;\leq\; |S| - 1.$$

So again, we satisfy the necessary conditions. Let us now compute how much we save.

$$\begin{aligned} -\sum_{(i,j)\in E} x_{ij} \lg \frac{d_j}{m} \;&=\; -\sum_{\substack{j \in V \\ d_j > 0}} \sum_{(i,j)\in E} x_{ij} \lg \frac{d_j}{m} \;=\; -\sum_{\substack{j \in V \\ d_j > 0}} \sum_{(i,j)\in E} \frac{1}{2d_j} \lg \frac{d_j}{m} \\ &=\; -\sum_{\substack{j \in V \\ d_j > 0}} d_j \frac{1}{2d_j} \lg \frac{d_j}{m} \;=\; -\sum_{\substack{j \in V \\ d_j > 0}} \frac{1}{2} \lg \frac{d_j}{m} \\ &=\; \frac{k}{2}\left(-\sum_{\substack{j \in V \\ d_j > 0}} \frac{1}{k} \lg \frac{d_j}{m} \right) \;\geq\; \frac{n}{2\alpha}\left(-\sum_{\substack{j \in V \\ d_j > 0}} \frac{1}{k} \lg \frac{d_j}{m} \right). \end{aligned} \tag{4}$$

Now, compare $-\sum_{j \in V : d_j > 0} \frac{1}{k} \lg \frac{d_j}{m}$ with

$$\mathcal{H}^{\text{in}}_{\text{deg}}(G) \;=\; -\sum_{\substack{j \in V \\ d_j > 0}} \frac{d_j}{m} \lg \frac{d_j}{m}.$$

(We often omit the $d_j > 0$ since we multiply by d_j, but here we include it to emphasise we are summing over the same indices.) We claim that

$$-\sum_{\substack{j \in V \\ d_j > 0}} \frac{1}{k} \lg \frac{d_j}{m} \;\geq\; -\sum_{\substack{j \in V \\ d_j > 0}} \frac{d_j}{m} \lg \frac{d_j}{m}. \tag{5}$$

Substituting into (4) implies that we save at least $\frac{n}{2\alpha}\mathcal{H}^{\text{in}}_{\text{deg}}(G)$, ignoring the $2n/\ln 2$ error term.

Note that the two terms in (5) are both weighted averages of $-\lg(d_j/m)$ for $j = 1, \ldots, k$. Suppose that

$$d_1 \geq \cdots \geq d_t \;\geq\; \frac{m}{k} \;\geq\; d_{t+1} \geq \cdots d_k.$$

Note that the terms $-\lg(d_1/m), \ldots, -\lg(d_t/m)$ are weighted more heavily in the right side of (5), whereas the terms $-\lg(d_{t+1}/m), \ldots, -\lg(d_k/m)$ are weighted more heavily in the left side of (5). However it is plain to see that *all* of the former terms are no larger than *all* of the latter terms. This implies the inequality we want, and the entire result. $\square$

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.

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

[BF10] Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In *Combinatorial Pattern Matching (CPM)*, page 138–150. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13509-5_13.

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

[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.

[CADS08] L. Castelli Aleardi, O. Devillers, and G. Schaeffer. Succinct representations of planar maps. *Theoretical Computer Science*, 408(2–3):174–187, November 2008. doi:10.1016/j.tcs.2008.08.016.

[CFJ08] Jean Cardinal, Samuel Fiorini, and Gwenaël Joret. Minimum entropy orientations. *Operations Research Letters*, 36(6):680–683, November 2008. doi:10.1016/j.orl.2008.06.010.

[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.

[DKPS10] Irit Dinur, Subhash Khot, Will Perkins, and Muli Safra. Hardness of finding independent sets in almost 3-colorable graphs. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, page 212–221, USA, 2010. IEEE Computer Society. doi:10.1109/FOCS.2010.84.

[Edm71] Jack Edmonds. Matroids and the greedy algorithm. *Math. Program.*, 1(1):127–136, 1971. doi:10.1007/BF01584082.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, July 1998. doi:10.1145/285055.285059.

[FFSG+20] Leo Ferres, José Fuentes-Sepúlveda, Travis Gagie, Meng He, and Gonzalo Navarro. Fast and compact planar embeddings. *Computational Geometry*, 89:101630, August 2020. doi:10.1016/j.comgeo.2020.101630.

[FM14] Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, June 2014. doi:10.1007/s00453-012-9664-0.

[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.

[GGV03] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, page 841–850, USA, 2003. SIAM.

[Gon85] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. URL: http://dx.doi.org/10.1016/0304-3975(85)90224-5, doi:10.1016/0304-3975(85)90224-5.

[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.

[GT05] Venkatesan Guruswami and Luca Trevisan. The complexity of making unique choices: Approximating 1-in- k SAT. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 99–110. Springer, 2005. doi:10.1007/11538462_9.

[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.

[INW25] Ziad Ismaili Alaoui, Namrata, and Sebastian Wild. Succinct preferential-attachment graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*. Springer, 2025. URL: https://www.wild-inter.net/publications/ismaili-alaoui-namrata-wild-2025, arXiv:2506.21436.

[Jac89] Guy Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 549–554, 1989. doi:10.1109/SFCS.1989.63533.

[JKK05] Norman L Johnson, Adrienne W Kemp, and Samuel Kotz. *Univariate discrete distributions*. John Wiley & Sons, 2005.

[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.

[Kam17] Shahin Kamali. Compact representation of graphs of small clique-width. *Algorithmica*, 80(7):2106–2131, September 2017. doi:10.1007/s00453-017-0365-6.

[KM23] Frank Kammer and Johannes Meintrup. Succinct planar encoding with minor operations. volume 283, pages 44:1–44:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.44.

[LMS19] 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.

[Mah08] Hosam M. Mahmoud. *Pólya Urn Models*. Chapman & Hall, 2008.

[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.

[MR01] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, January 2001. doi:10.1137/s0097539799364092.

[MSOI+02] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, October 2002. URL: http://dx.doi.org/10.1126/science.298.5594.824, doi:10.1126/science.298.5594.824.

[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, March 2014. doi:10.1016/j.jda.2013.07.004.

[NS14] Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 10(3):1–39, may 2014. doi:10.1145/2601073.

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

[PHS+22] Jeremy Paton, Harrison Hartle, Huck Stepanyants, Pim van der Hoorn, and Dmitri Krioukov. Entropy of labeled versus unlabeled networks. *Physical Review E*, 106(5), November 2022. doi:10.1103/physreve.106.054308.

[RRRS07] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4):43–es, nov 2007. doi:10.1145/1290672.1290680.

[SG18] Wojciech Szpankowski and Ananth Grama. Frontiers of science of information: Shannon meets turing. *Computer*, 51(1):28–38, January 2018. doi:10.1109/mc.2018.1151004.

[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. URL: https://www.wild-inter.net/publications/tsakalidis-wild-zamaraev-2023, arXiv:2010.04108, doi:10.1007/s00453-022-01039-2.

Appendix

A. Hardness of Hitting Set

In this section, we walk through the proof of [GT05] for proving the hardness of (α, r, k)-APX-EHS for every $\alpha < e$ and for some fixed r, k depending only on α. We first give an equivalent formulation of this problem as a CSP.

Definition A.1: *In (α, r, k)-APX-EHS, we are given an r-regular k-uniform hypergraph H. A solution is a set S of vertices. The value of a solution is the proportion of edges T such that $|S \cap T| = 1$. We must decide whether there exists a solution with value 1, or not even a solution with value $1/\alpha$.* ◁

To reiterate, [GT05] proves that for every $\alpha < e$ there exists k such that this problem is NP-hard *but without the r-regularity condition.* However, their reduction implicitly produces an r-regular instance, where r depends only on α. In this section we will merely show this fact, going through the reduction, without reproving other facts.

The reduction of [GT05] starts from the p-prover proof system of Feige [Fei98], used to prove hardness results for set cover. Let us first define this proof system.

Definition A.2: *Fix constants $p \geq 2$ and u, an even integer. An instance of size n of the p-prover system is defined as follows. Let $R = (5n)^u$, $Q = n^{u/2}(5n/3)^{u/2}$, $A = 2^u$, $B = 4^u$. The instance is a hypergraph $H = (V, E)$, where each edge is equipped with a sequence of functions from $[B]$ to $[A]$, with the following properties.*

Uniformity. *The hypergraph is p-uniform and p-partite, i.e. the vertex set is given by $V = Q_1 \cup \ldots \cup Q_p$, and each edge takes exactly one vertex from each part Q_i.*

Regularity. *The hypergraph is regular with regularity*

$$\frac{R}{Q} \;=\; \frac{(5n)^u}{n^{u/2}(5n/3)^{u/2}} \;=\; 15^{u/2}.$$

Vertices. *Each part Q_i has size $|Q_i| = Q$.*

Edges. *There are R edges in the hypergraph. Edge $r \in [R]$ is denoted by $(q_{r1}, \ldots, q_{rp})$, where $q_{ri} \in Q_i$.*

Projections. *Each edge $(q_{r1}, \ldots, q_{rp})$ is equipped with a sequence of projections $(\pi_{r1}, \ldots, \pi_{rp})$, where each π_{ri} is a function from $[B]$ to $[A]$ that is $(B/A) = 2^u$-to-1, i.e. the preimage through π_{ri} of every value has size 2^u.*

A solution to this problem is a labelling $a : V \to [B]$. Such a solution:

- Strongly satisfies *edge r if $\pi_{ri}(a(q_{ri})) = \pi_{rj}(a(q_{rj}))$ for* all $i, j \in [p]$.
- Weakly satisfies *edge r if $\pi_{ri}(a(q_{ri})) = \pi_{rj}(a(q_{rj}))$ for* at least one $i, j \in [p]$ *with $i \neq j$.* ◁

Feige then proves the following key NP-hardness result.

Theorem A.3: *For some universal constant $0 < c < 1$, for every $p \geq 2$ and for u large enough, the following problem is NP-hard. Given an instance of the p-prover system with parameter u, it is NP-hard to distinguish whether:*

Yes instance. *There exists a labelling that strongly satisfies all edges.*

No instance. *No labelling weakly satisfies even a p^2c^u fraction of the edges.* ◁

To make this notion more intuitive, we explain why it is called a *proof system*. Imagine that each part $i \in [p]$ corresponds to one of p provers, which are trying to prove some fact to a checker. Each node $q_i \in Q_i$ is a query that the checker could ask prover i. A solution is a strategy for answering these queries. The checker selects an edge of the hypergraph at random, and asks the corresponding queries to each prover. The checker then passes these answers through the projection functions for the given edge. They are then strongly convinced of the correctness of the proof if the answer is unanimous; and weakly convinced if at least two of the answers are equal.

Feige's hardness result essentially says the following: for any instance of e.g. 3-SAT, it is possible, in polynomial time, to create a p-prover system where (i) if the 3-SAT instance is satisfiable then there exists a proof that strongly convinces the checker with probability 1, and (ii) if the 3-SAT instance is unsatisfiable, then no proof even weakly convinces the checker with probability p^2c^u.

We now give the reduction of [GT05], where we trace the r-regularity, as well.

Theorem A.4: *Fix $\varepsilon > 0$, and let $1/\alpha = 1/e + \varepsilon$. There exists a positive integer p such that for all large enough u, the following holds. Let $k = 4^u$ and $r = 15^u p^{2^u-1}$. Given an instance of the p-prover proof system with parameter u, we can create in polynomial time an instance of (α, r, k)-APX-EHS that is a Yes resp. No instance whenever the p-prover proof system we were given was a Yes resp. No instance. Hence, (α, r, k)-APX-EHS is NP-hard.* ◁

Proof: Suppose that the input p-prover system with parameter u is denoted by the same symbols as in Definition A.2. We must now output an r-regular k-uniform hypergraph with the required properties. The hypergraph we output is constructed as follows.

Vertices. For every $i \in [p]$, $q \in Q_i$ and $b \in [B]$, we introduce (i, q, b) as a vertex.

Edges. For every $r \in [R]$ and $\mathbf{x} = (x_1, \ldots, x_A) \in [p]^A$, we introduce an edge

$$S_{r\mathbf{x}} \;=\; \{(i, q_{ri}, b) \mid x_{\pi_{ri}(b)} = i\}.$$

The soundness and completeness of this reduction is proved by [GT05] – hence if the original instance was a yes/no-instance of the p-prover game with parameter u, this hypergraph is a yes/no-instance of (α, r, k)-APX-EHS. What remains is to show k-uniformity and r-regularity; the first is also shown by [GT05], but we give the proof again for completeness.

Uniformity. We must count the size of each set $S_{r\mathbf{x}}$. Let us fix the *output* of $\pi_{ri}(b)$ in the definition of this set. Then:

$$S_{r\mathbf{x}} \;=\; \bigcup_{a \in [A]} \{(i, q_{ri}, b) \mid i = x_a, \pi_{ri}(b) = a\}.$$

Now note that for every fixed a, we may select b in $B/A = 2^u$ ways in order to make $\pi_{ri}(b) = a$, since π_{ri} is a 2^u-to-1 function. Then, i is fixed as a function of a and $\mathbf{x}$, and likewise q_{ri} is fixed as a function of r and i. So, we see that for each choice of $a \in [A]$, $S_{r\mathbf{x}}$ contains precisely 2^u different values. It follows that $|S_{r\mathbf{x}}| = |A|2^u = 4^u = k$.

Regularity. We must count how many times any fixed vertex (i, q, b), where $q \in Q_i$ and $b \in [B]$, belongs to an edge $S_{r\mathbf{x}}$. For this to be the case, we first need that $q = q_{ri}$. This holds for exactly $15^{u/2}$ choices of r, since the original p-prover system instance is $15^{u/2}$-regular. Now, fix one such choice of r and assume $q = q_{ri}$. Let us count for how many vectors $\mathbf{x} = (x_1, \ldots, x_A) \in [p]^A$ we have $(i, q_{ri}, b) \in S_{r\mathbf{x}}$. The only condition for this to be the case is that $x_{\pi_{ri}(b)} = i$. Since we have fixed (i, q_{ri}, b) and r, the index $\pi_{ri}(b)$ is fixed. Hence there are exactly p^{A-1} choices of $\mathbf{x}$ that lead to $(i, q_{ri}, b) \in S_{r\mathbf{x}}$. Overall, we see that the instance is regular with regularity

$$15^u p^{A-1} \;=\; 15^u p^{2^u - 1} \;=\; r.$$

This concludes the proof. □

B. A Tree is Never Worse Than a Forest

One may wonder whether a *strictly* lower indegree entropy can be achieved by removing a spanning non-tree forest instead of a tree. This section answers this question with a *No*; there is always a spanning tree whose elimination from the graph produces an optimal indegree entropy. Indeed, we show that removing an additional edge never increases $H(G)$, and strictly decreases $H(G)$ (unless $H(G) = 0$ already).

First of all, let us prove a simple arithmetic claim.

Lemma B.1: *For $a, b \in \mathbb{R}_{\geq 1}$ and $a \geq b \geq 1$, we have*

$$(a-1)\lg(a-1) - (b-1)\lg(b-1) \;\leq\; a\lg a - b\lg b,$$

with equality only for $a = b$. ◁

Proof: After dividing by $\ln 2$ and rearranging, it suffices to show that

$$f(x) \;=\; (x-1)\ln(x-1) - x\ln x$$

is a strictly decreasing function for $x \in \mathbb{R}_{>1}$. The derivative $f'(x)$ is computed as:

$$f'(x) \;=\; \frac{\mathrm{d}}{\mathrm{d}x}(x-1)\ln(x-1) - x\lg x \;=\; \ln(x-1) - \ln(x) \;=\; \ln(1 - 1/x),$$

which is $f'(x) < 0$ for any $x > 1$. □

Now, we are ready to show the main claim of this section. Let $\mathcal{F}(G)$ and $\mathcal{T}(G) \subseteq \mathcal{F}(G)$ denote the set of spanning forests and spanning trees of G, respectively.

Lemma B.2: *Let $G = (V, E)$ be a weakly connected, directed graph. Then there exists $T \in \mathcal{T}(G)$ such that*

$$H(G - T) \;=\; \min_{F \in \mathcal{F}(G)} H(G - F).$$

◁

Proof: We actually show the stronger claim hinted at above: apart from the degenerate case of a star graph with indegree entropy 0, removing any edge strictly improves the resulting indegree entropy. Formally, recall that $H(G) = -\sum_{v \in V} d_v \lg(\frac{d_v}{m})$. We show that $H(G - e) \leq H(G)$ for any $e \in E$, with equality only for $H(G) = 0$. This immediately proves the claim since spanning trees are precisely the maximal spanning forests of G.

We rewrite the indegree entropy for G as

$$H(G) \;=\; -\sum_{v\in V} d_v \lg\!\left(\frac{d_v}{m}\right) \;=\; -\sum_{v\in V} d_v \lg d_v + m\lg m.$$

Removing $e=(u,w)$ from G makes exactly one indegree smaller by 1: d_w decreases upon deleting e from G. We have

$$H(G-e) \;=\; -\sum_{v\in V\setminus\{w\}} d_v \lg d_v \;-\; (d_w-1)\lg(d_w-1) \;+\; (m-1)\lg(m-1),$$

where d_w is the indegree of w in G (before deleting e). Comparing with

$$H(G) \;=\; -\sum_{v\in V\setminus\{w\}} d_v \lg d_v \;-\; d_w \lg d_w \;+\; m\lg m$$

yields that $H(G-e)<H(G)$ if and only if

$$(m-1)\lg(m-1)-(d_w-1)\lg(d_w-1) \;<\; m\lg m - d_w\lg d_w \tag{6}$$

Since $1\le d_w\le m$, we can apply Lemma B.1 with $a=m$ and $b=d_w$ to obtain (6), with equality only for $m=d_w$, i.e. when in G all edges point to w. □

C. Copy Model

In this section, we describe the copy model for generating random graphs studied in [TMS20]. It is loosely motivated by protein interaction networks, in which mutations of the gene coding for a known protein creates a new, but functionally equivalent protein, which then shares the interaction relations of its ancestor. In the random copy model of [TMS20], we start with a seed graph G_0 and repeatedly copy a vertex chosen uniformly at random among the current graph, thus creating a twin of the copied vertex. Turowski et al. compute the entropy of the resulting graph distribution, both for the labelled and unlabelled case, and describe corresponding compression methods, but without query support. We show that our TREX data structure with twin removal yields on asymptotically *instance-optimal* representation of unlabelled graphs from the copy model that includes full support of efficient navigational operations.

C.1. The Random Copy Model

We recall the definitions and notation from [TMS20]. (We match this to our notation from Section 5 below.)

The random copy model is parameterized by an initial graph G_0 on n_0 vertices. For each step $1\le i\le n$, the graph G_i is constructed from G_{i-1} as follows:

1. Select a vertex $v\in V(G_{i-1})$ uniformly at random.
2. Add a new vertex v_i to the graph.
3. Connect v_i to all neighbors of v.

Note that v and v_i themselves are not connected.

After n steps, the graph G_n has n_0+n vertices. The structure of G_n depends strongly on the choice of G_0. The original vertices of G_0 are denoted $U=V(G_0)=\{u_1,\dots,u_{n_0}\}$, and the vertices added in the duplication process are denoted $V=\{v_1,\dots,v_n\}$. Moreover, we denote by $N_n(v)$ the neighbourhood of vertex v in G_n, that is, all vertices that are adjacent to v in G_n.

Note that Turowski et al. considered undirected graphs, but all the above definitions can be easily generalised to directed graphs.

C.2. Properties

We now define the concepts of parent and ancestor of a vertex. A vertex w is called the *parent* of v if v was copied from w at some step $1 \leq i \leq n$. By repeatedly moving from v to its parent, we eventually reach a vertex $u \in V(G_0)$; this vertex u is called the *ancestor* of v. Conversely, we define the set of *descendants* of a vertex $u_i \in V(G_0)$ in G_n as the set of all vertices $v \in V(G_n)$ whose ancestor is u_i. For $i = 1, \ldots, n_0$, we denote the descendants of u_i by $\mathcal{C}_{i,n}$.

The key observation about the copy model is the invariant that all descendants of a vertex u_i are *twins*. This is initially true when a new twin is added to the graph, and later additions of twins cannot separate a vertex v and its ancestor u since cloning any neighbour of v resp. u would also include u resp. v; (see also Lemma 1 of Turowski et al. [TMS20]).

Let $C_{i,n} := |\mathcal{C}_{i,n}|$, that is, the number of vertices in G_n that are the twins of u_i (including u_i itself). It is not hard to see that the sequence of variables $(C_{i,n})_{i=1}^{n_0}$ can be described as an urn model (indeed the classical Pólya urn [Mah08, JKK05]): The urn contains balls of n_0 different "colors" $u_1, \ldots, u_{n_0}$. At time $n = 0$, we have exactly one ball per color for a total of n_0 balls. In each step, we pick a ball uniformly at random from the urn and return both the chosen ball, as well as one *additional* ball of the same color to the urn. The joint distribution of "new" color multiplicities $(C_{i,n} - 1)_{i=1}^{n_0}$ is then Dirichlet-multinomial $\mathrm{DirMult}(n; \alpha_1, \ldots, \alpha_{n_0})$ distributed with parameters $\alpha_i = 1$ for $1 \leq i \leq n_0$ (the initial multiplicities of colors in the urn). $\mathrm{DirMult}(n; 1, \ldots, 1)$ is a *uniform* distribution over all vectors that sum to n:

$$\mathbb{P}\big[(C_{i,n})_{i=1}^{n_0} = (k_i + 1)_{i=1}^{n_0}\big] \;=\; \begin{cases} \dfrac{1}{\binom{n+n_0-1}{n}}, & \text{if } \displaystyle\sum_{i=1}^{n_0} k_i = n, \\ 0, & \text{otherwise.} \end{cases} \qquad \forall 1 \leq i \leq n_0 : k_i \in \mathbb{N}_{\geq 1} \tag{7}$$

C.3. Entropy

Under the assumption that the initial graph G_0 is fixed/known and *asymmetric*, i.e. its automorphism group is of size 1, Turowski et al. [TMS20] determined the asymptotic entropy for both unlabeled and labelled random graphs generated by this model. They proved [TMS20, Theorem 2] that the entropy of *labeled* graphs is

$$(H_{n_0} - 1)\lg(e) \cdot n \;+\; \frac{n_0 - 1}{2} \cdot \lg n \;\pm\; O(n_0 \log n_0), \qquad \text{with } H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} \tag{8}$$

while the entropy of the unlabelled graph (the entropy of the isomorphism class of a random graph from the model, a.k.a. the "structural entropy") is significantly smaller [TMS20, Theorem 1]:

$$(n_0 - 1)\lg n \;\pm\; O(n_0 \log n_0).$$

Note that for constant $n \to \infty$, the labelled graph entropy grows exponentially faster than the unlabelled one ($\Theta(n)$ vs. $\Theta(\log n)$); the copy model is hence an extreme case where the vast majority of information in a (labelled) graphs lies in the vertex labels, not the underlying graph structure.

In [TMS20, Theorems 4, 5], they also provide optimal algorithms for compressing both unlabeled and labelled graphs generated by the full copy model. It is based on arithmetic coding and is optimal up to two bits. This is provided, we have already compressed G_0. This matches the entropies up to a constant additive term. Neither representation is known to support efficient queries.

C.4. TREX with twin removal is instance-optimal

Beyond the entropy of the distribution, from (7) it is indeed easy to compute the instance-specific information content $\lg(1/\mathbb{P}[G \mid G_0])$ of a graph G drawn according to the copy model with seed graph G_0 (again assuming G_0 is asymmetric). Given G_0 and n, the probability of a graph G to arise in the copy model is $\mathbb{P}[G \mid G_0, n] = 1/\binom{n+n_0-1}{n}$, provided the twin reduction of G is G_0 (otherwise, $\mathbb{P}[G \mid G_0, n] = 0$). In the former case, we have

$$\begin{aligned} \lg(1/\mathbb{P}[G \mid G_0, n]) &= \lg \binom{n+n_0-1}{n_0-1} \\ &= (n_0-1)\lg\left(\frac{n+n_0-1}{n_0-1}\right) + O(n_0) \\ &= (n_0-1)\lg(n+n_0-1) \pm O(n_0 \log n_0) \end{aligned} \tag{9}$$

Note that in our notation from Section 5, the overall graph size is $N = n + n_0$ and the *seed graph* G_0 in the copy model is the *twin reduction* of G in twin removal. In Section 5, we denoted the size of the twin reduction by $n = |V(G_0)|$ instead of n_0 here. The *representative* of the twin class in Section 5 is called the *ancestor* of a vertex in G here.

With this, we find that the claimed space in Theorem 5.1 *almost* matches $\lg(1/\mathbb{P}[G \mid G_0, n])$ – but not quite! However, a closer look at bitvector B reveals that its first bit is *always* 1, and we thus need not encode it at all. The bits we actually have to encode are only $N-1$ bits, of which exactly $n-1$ are 1. The space for B thus becomes $\lg\binom{N-1}{n-1} \leq (n-1)\lg(N-1) + o(N)$, which is asymptotically for large N the instance-optimal space usage to represent the unlabeled graph G given G_0 (which we encode using TREX).