Compression of Graphical Structures: Fundamental Limits, Algorithms, and Experiments [*]

December 11, 2009

Yongwook Choi and Wojciech Szpankowski[†]
Department of Computer Science
Purdue University
W. Lafayette, IN 47907
U.S.A.
ywchoi@purdue.edu, spa@cs.purdue.edu

Abstract

Information theory traditionally deals with "conventional data," be it textual data, image, or video data. However, databases of various sorts have come into existence in recent years for storing "unconventional data" including biological data, social data, web data, topographical maps, and medical data. In compressing such data, one must consider two types of information: the information conveyed by the *structure itself*, and the information conveyed by the data labels implanted in the structure. In this paper, we attempt to address the former problem by studying information of graphical structures (i.e., unlabeled graphs). As the first step, we consider the Erdős-Rényi graphs $\mathcal{G}(n,p)$ over n vertices in which edges are added randomly with probability p. We prove that the *structural entropy* of $\mathcal{G}(n,p)$ is

$$\binom{n}{2}h(p) - \log n! + o(1) = \binom{n}{2}h(p) - n\log n + O(n),$$

where $h(p) = -p\log p - (1-p)\log(1-p)$ is the entropy rate of a conventional memoryless binary source. Then, we propose a two-stage compression algorithm that asymptotically achieves the structural entropy up to the first two leading terms. Our algorithm runs in $O(n+e)$ time on average where e is the number of edges. To the best of our knowledge, this is the first provable (asymptotically) optimal graph compressor. We use combinatorial and analytic techniques such as generating functions, Mellin transform, and poissonization to establish these findings. Our experiments confirm theoretical results and show the usefulness of our algorithm for real-world graphs such as the Internet, biological networks, and social networks.

Index Terms: Graph automorphism, structural entropy, Erdős-Rényi graphs, arithmetic encoder, graph compressor, digital trees, poissonization, Mellin transform, analytic information theory.

[*]A preliminary version of this work was presented at the 2009 ISIT, Seoul, S. Korea.

[†]This work was supported in part by the NSF Grants DMS-0800568 and CCF-0830140, NSA Grant H98230-08-1-0092, and the AFOSR Grant FA8655-08-1-3018.

1 Introduction

In 1948 Shannon introduced a metric for information launching the field of information theory. However, as observed by Brooks [5] and others [29, 39], there is no theory that gives us a metric for information embodied in structure. Shannon himself in his 1953 little known paper [35] argued for an extension of information theory to "non-conventional data" (i.e., lattices). Indeed, data is increasingly available in various forms (e.g., sequences, expressions, interactions, structures) and in exponentially increasing amounts. For example, in biology large amounts of data are now in public domain on gene regulation, protein interactions, and metabolic pathways. Most of such data is multidimensional and context dependent. Therefore, it necessitates novel theory and efficient algorithms for extracting meaningful information from non-conventional data structures. Typically, a data file of this new type (e.g., biological data, topographical maps, medical data, volumetric data) is a "data structure" conveying a "shape" and consisting of labels implanted in the structure. In understanding such data structures, one must take into account two types of information: the information conveyed by the structure itself and the data labels implanted in the structure. In this paper, we address the former problem in order to quantify the amount of information in networks such as the Internet, social networks, biological networks, and economic networks.

Unconventional data contains more sophisticated structural relations. For example, a graph can be represented by a binary matrix that further can be viewed as a binary sequence. However, such a string does not exhibit internal symmetries that are conveyed by the so-called *graph automorphism* (making certain sequences/matrices are "indistinguishable"). The main challenge in dealing with such structural data is to identify and describe these structural relations. In fact, these "regular properties" constitute "useful (extractable) information" understood in the spirit of Rissanen "learnable information" [30] (cf. also [25, 26]).

As the first step in understanding structural information, we restrict our attention to structures on graphs. More specifically, we study *unlabeled graphs* (or structures) generated by a memoryless source known as the Erdős-Rényi model [4] in which edges are added randomly with probability p. This model induces a probability distribution on structures so that one can compute Shannon entropy giving us a fundamental limit on lossless unlabeled graph compression. We prove that this *structural entropy* $H_{\mathcal{S}}$ is

$$\binom{n}{2}h(p) - \log n! + o(1) = \binom{n}{2}h(p) - n\log n + O(n),$$

where n is the number of vertices and $h(p) = -p\log p - (1-p)\log(1-p)$ is the entropy rate of a conventional memoryless binary source.[1] In addition, we prove that, for almost every structure S from this model, the probability of S is very close to $2^{-H_{\mathcal{S}}}$ for large n, which is a manifestation of AEP (asymptotic equipartition property) for the Erdős-Rényi graphs.

Then we design and analyze a graphical (structure) compression algorithm, called Szip, that asymptotically achieves the compression rate

$$\binom{n}{2}h(p) - n\log n + O(n)$$

that matches the lower bound up to the first two leading terms with high probability. Our algorithm consists of two stages. It first encodes a structure into two binary strings that

[1] All logarithms are to the base 2 throughout this paper.

are then compressed using an arithmetic encoder. Our algorithm runs in $O(n+e)$ time on average, where e is the number of edges. This is faster than $O(n^2)$-time algorithms, discussed in [7, 27], theoretically as well as in practice since most real-world graphs are very sparse. Experimental results on both real-world networks and the Erdős-Rényi graphs confirm the efficiency and utility of our algorithm.

There are other possible metrics of information content of a graph. For example, "topological entropy" discussed in [29, 39] attempts to characterize the distinctiveness of vertex degrees by partitioning all vertices into subsets of the same long term connectivity. As a by-product of our analysis, we prove in this paper that such topological entropy is equal to $\log n + o(1)$ for the Erdős-Rényi random graph model. Furthermore, the most popular "graph entropy" is due to Körner who generalized standard Shannon entropy to "undistinguished symbols" [36]. Graph entropy is a function of the graph and a probability distribution on the vertices. Roughly speaking it reflects the number of bits you need to transmit to describe the vertex when you want to distinguish only between vertices that are connected (connected vertices represent "distinguishable symbols"). For example, if the graph is complete, then one must distinguish between any two vertices. In this case, the Körner entropy achieves the highest value that coincides with the Shannon entropy. But a complete graph has the simplest structure to describe, thus it should be clear that our structural entropy is quite different than the Körner graph entropy.

Literature on graphical structure compression is scarce. In 1984, Turan [40] raised the question of finding efficient coding method for general unlabeled graphs on n vertices, suggesting a lower bound of $\binom{n}{2} - n\log n + O(n)$ bits. In 1990, Naor [27] proposed such a representation that is optimal up to the first two leading terms when all unlabeled graphs are equally likely. Naor's result is asymptotically a special case of ours when $p = 1/2$. Finally, in a recent paper Kieffer et al. [22] presented a structural complexity of a binary tree, in a spirit similar to ours. There also have been some heuristic methods for real-world graphs compression including Adler and Mitzenmacher [1] (see also [6]), who proposed an encoding technique for web graphs, and similar idea has been used in [37] for compressing sparse graphs. Recently, attention has been paid to grammar compression for some data structures: Peshkin [28] proposed an algorithm for a graphical extension of the one-dimensional SEQUITUR compression method. However, SEQUITUR is known not to be asymptotically optimal [32]. Therefore, the Peshkin method already lacks asymptotic optimality in the 1D case. To the best of our knowledge our algorithm is the first provable asymptotically optimal compression scheme for graphical structures.

The paper is organized as follows. The structural entropy of a graph is defined in Section 2 and compared to the conventional graph entropy. Our algorithm is described in Section 3, where we derive the structural entropy for $\mathcal{G}(n,p)$. We also present there our experimental results. Our main results are proved in Sections 4 and 5, where we introduce random binary trees that resemble tries and digital search trees. We use analytic techniques such as generating functions, Mellin transform, and poissonization to establish our results.

2 Structural Entropy

In this section, we formally define the structural entropy of a random (unlabeled) graph model. Given n distinguishable vertices, a random graph is generated by adding edges randomly. This random graph model $\mathcal{G}$ produces a probability distribution on graphs, and the graph entropy $H_{\mathcal{G}}$ is defined naturally as

$$H_{\mathcal{G}} = \mathbf{E}[-\log P(G)] = -\sum_{G \in \mathcal{G}} P(G) \log P(G),$$

where $P(G)$ is the probability of a graph G. We now introduce a *random structure model* $\mathcal{S}$ for the unlabeled version of a random graph model $\mathcal{G}$. In such a model, graphs are generated in the same manner as in $\mathcal{G}$, but they are thought of as unlabeled graphs. That is, the vertices are indistinguishable, and the graphs having "the same structure" are considered to be the same even if their labeled versions are different. Thus, we shall use the terms *unlabeled graphs* and *structures* interchangeably. For a given structure $S \in \mathcal{S}$, the probability of S can be computed as

$$P(S) = \sum_{G \cong S, G \in \mathcal{G}} P(G).$$

Here $G \cong S$ means that G and S have the same structure, that is, S is *isomorphic* to G. If all isomorphic labeled graphs have the same probability, then for any labeled graph $G \cong S$,

$$P(S) = N(S) \cdot P(G), \tag{1}$$

where $N(S)$ is the number of different labeled graphs that have the same structure as S. The *structural entropy* $H_{\mathcal{S}}$ of a random graph $\mathcal{G}$ can be defined as the entropy of a random structure $\mathcal{S}$,

$$H_{\mathcal{S}} = \mathbf{E}[-\log P(S)] = -\sum_{S \in \mathcal{S}} P(S) \log P(S),$$

where the summation is over all distinct structures.

Example: In Figure 1(a), we draw different graphs built on three vertices. Let us assume that they are equally probable in some random graph model, that is, $P(G_i) = 1/8$ for $1 \le i \le 8$. Then the entropy of this random graph $\mathcal{G}$ is $H_{\mathcal{G}} = -8 \cdot \frac{1}{8} \log \frac{1}{8} = 3$. Let $\mathcal{S}$ be the random structure that corresponds to $\mathcal{G}$. In Figure 1(b), we present all different structures that can be generated by $\mathcal{S}$. Since $N(S_1) = N(S_4) = 1$ and $N(S_2) = N(S_3) = 3$, thus $P(S_1) = P(S_4) = 1/8$ and $P(S_2) = P(S_3) = 3/8$. The entropy of random structure $\mathcal{S}$ is $H_{\mathcal{S}} = -2 \cdot \frac{1}{8} \log \frac{1}{8} - 2 \cdot \frac{3}{8} \log \frac{3}{8} \approx 1.811$. ∎

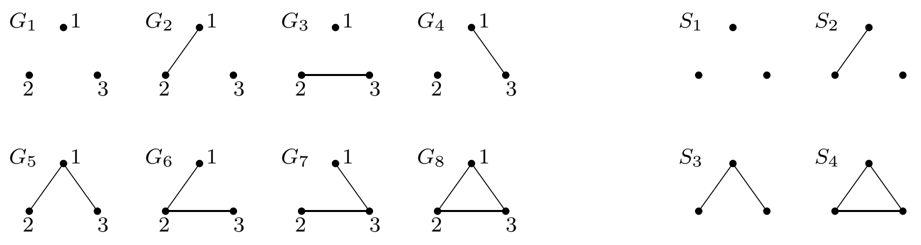


Figure 1: All different graphs and structures built on three vertices.

In order to compute the probability of a given structure S, one needs to estimate the number of ways to construct a given structure S, denoted as $N(S)$. For this, we need to consider the automorphisms of a graph. An *automorphism* of a graph G is an adjacency preserving permutation of vertices of G. The collection $\mathrm{Aut}(G)$ of all automorphisms of G is called *the automorphism group* of G. In the sequel, $\mathrm{Aut}(S)$ of a structure S denotes $\mathrm{Aut}(G)$ for some labeled graph G such that $G \cong S$. In group theory, it is well known that [14, 15]

$$N(S) = \frac{n!}{|\mathrm{Aut}(S)|}. \tag{2}$$

We also easily observe that $1 \le |\mathrm{Aut}(S)| \le n!$.

Example: In Figure 2(a), the graph G has exactly four automorphisms, that is, in the usual cyclic permutation representation: $(v_1)(v_2)(v_3)(v_4)$, $(v_1)(v_4)(v_2v_3)$, $(v_1v_4)(v_2)(v_3)$, and $(v_1v_4)(v_2v_3)$. For example, $(v_1)(v_4)(v_2v_3)$ stands for a permutation π such that $\pi(v_1) = v_1$, $\pi(v_4) = v_4$, $\pi(v_2) = v_3$, and $\pi(v_3) = v_2$. Thus, by (2), G has $4!/4 = 6$ different labeling as shown in Figure 2(b). ∎

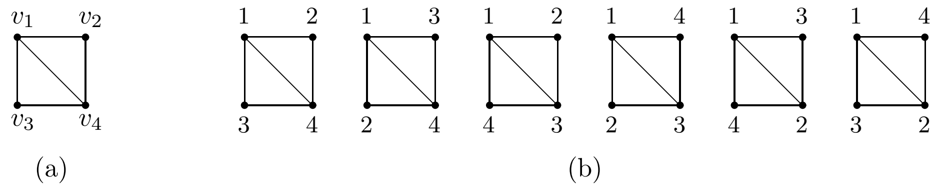


Figure 2: The six different labeling of a graph.

With these preliminary definitions, we are now in the position to present a relationship between $H_{\mathcal{G}}$ and $H_{\mathcal{S}}$.

Lemma 1 *If all isomorphic graphs have the same probability, then*

$$H_{\mathcal{S}} = H_{\mathcal{G}} - \log n! + \sum_{S \in \mathcal{S}} P(S) \log |\mathrm{Aut}(S)|,$$

for any random graph $\mathcal{G}$ and its corresponding random structure $\mathcal{S}$, where $\mathrm{Aut}(S)$ *is the automorphism group of S.*

Proof: Observe that for any $\mathcal{G}$ and $\mathcal{S}$

$$\begin{aligned}
H_{\mathcal{G}} &= -\sum_{G \in \mathcal{G}} P(G) \log P(G) \\
&= -\sum_{S \in \mathcal{S}} \sum_{G \cong S, G \in \mathcal{G}} P(G) \log P(G) \\
&= -\sum_{S \in \mathcal{S}} \sum_{G \cong S, G \in \mathcal{G}} \frac{P(S)}{N(S)} \log \frac{P(S)}{N(S)} \qquad \text{(by (1))} \\
&= -\sum_{S \in \mathcal{S}} N(S) \cdot \frac{P(S)}{N(S)} \log \frac{P(S)}{N(S)} \\
&= H_{\mathcal{S}} + \sum_{S \in \mathcal{S}} P(S) \log \frac{n!}{|\mathrm{Aut}(S)|} \qquad \text{(by (2))} \\
&= H_{\mathcal{S}} + \log n! - \sum_{S \in \mathcal{S}} P(S) \log |\mathrm{Aut}(S)|.
\end{aligned}$$

This proves the lemma. ∎

The last term of the structural entropy $\sum_{S\in\mathcal{S}} P(S)\log|\mathrm{Aut}(S)|$ can vary from 0 to $n\log n$ since $1 \le |\mathrm{Aut}(S)| \le n!$. However, as we shall see in most random graph models there is not much symmetry, and hence $\sum_{S\in\mathcal{S}} P(S)\log|\mathrm{Aut}(S)| = o(1)$. In order to develop further the idea of information in a random structure, hereafter we will focus on the Erdős-Rényi random graph [4].

3 Main Results

In this section, we first compute the structural entropy for the Erdős-Rényi random graph. As it is well known, such entropy constitutes a lower bound for lossless compression. Then we describe our optimal compression algorithm that asymptotically achieves this lower bound up to the second leading term with high probability. Finally, we present experimental results to show the efficiency and utility of our algorithm.

3.1 Structural Entropy of the Erdős-Rényi Model

In the Erdős-Rényi random graph model $\mathcal{G}(n,p)$, graphs are generated randomly on n vertices with edges chosen independently with probability $0 < p < 1$. If a graph G in $\mathcal{G}(n,p)$ has k edges, then

$$P(G) = p^k q^{\binom{n}{2}-k},$$

where $q = 1-p$. Let $\mathcal{S}(n,p)$ be the random structure model (unlabeled graphs) corresponding to $\mathcal{G}(n,p)$. Then, by (1) if $S \in \mathcal{S}(n,p)$ has k edges,

$$P(S) = N(S)\cdot p^k q^{\binom{n}{2}-k}.$$

To compute the entropy of $\mathcal{S}(n,p)$ we need to estimate $N(S)$. For this, we must study an important property of $\mathcal{S}(n,p)$ (or equivalently, $\mathcal{G}(n,p)$), namely *asymmetry.* A graph is said to be *asymmetric* if its automorphism group does not contain any permutation other than the identity (i.e., $(v_1)(v_2)\cdots(v_n)$); otherwise it is called *symmetric.* It is known that almost every graph from $\mathcal{G}(n,p)$ is asymmetric [11, 23]. In the sequel, we write $a_n \ll b_n$ to mean $a_n = o(b_n)$ when $n\to\infty$. For completeness, we present in Appendix A a slightly generalized proof of Kim et al.'s result [23].

Lemma 2 (Kim, Sudakov, and Vu, 2002) *For all p satisfying $\frac{\ln n}{n} \ll p$ and $1-p \gg \frac{\ln n}{n}$ (for connected graphs), a random graph $G\in\mathcal{G}(n,p)$ is symmetric with probability $O\left(n^{-w}\right)$ for any positive constant $w > 1$.*

Using this property, we next present the structural entropy of $\mathcal{G}(n,p)$ and establish the asymptotic equipartition property (AEP), that is, typical probability of a structure S.

Theorem 1 *For large n and all p satisfying $\frac{\ln n}{n} \ll p$ and $1-p \gg \frac{\ln n}{n}$, the following holds:*
(i) *The structural entropy $H_{\mathcal{S}}$ of $\mathcal{G}(n,p)$ is*

$$H_{\mathcal{S}} = \binom{n}{2}h - \log n! + O\left(\frac{\log n}{n^{\alpha}}\right), \qquad \alpha > 0,$$

(ii) *(AEP) For a structure $S \in \mathcal{S}(n,p)$ and $\epsilon > 0$,*

$$P\left(\left|-\frac{1}{\binom{n}{2}}\log P(S) - h + \frac{\log n!}{\binom{n}{2}}\right| < \epsilon\right) > 1 - 2\epsilon, \tag{3}$$

where $h := h(p) = -p\log p - (1-p)\log(1-p)$ is the entropy rate of a binary memoryless source.

Proof: Let us first compute the entropy $H_{\mathcal{G}}$ of $\mathcal{G}(n,p)$. In $\mathcal{G}(n,p)$, $m = \binom{n}{2}$ distinct edges are independently selected with probability p, and thus there are 2^m different labeled graphs. That is, each graph instance can be considered as a binary sequence X of length m. Thus,

$$H_{\mathcal{G}} = -\mathbf{E}[\log P(X_1^m)] = -m\mathbf{E}[\log P(X_1)] = \binom{n}{2}h.$$

By Lemma 1,

$$H_{\mathcal{S}} = \binom{n}{2}h - \log n! + A$$

where

$$A = \sum_{S\in\mathcal{S}} P(S)\log|\mathrm{Aut}(S)|.$$

Now we show that $A = o(1)$ to prove part (i).

$$\begin{aligned}
A &= \sum_{S\in\mathcal{S}(n,p)\text{ is symmetric}} P(S)\log|\mathrm{Aut}(S)| + \sum_{S\in\mathcal{S}(n,p)\text{ is asymmetric}} P(S)\log|\mathrm{Aut}(S)| \\
&= \sum_{S\in\mathcal{S}(n,p)\text{ is symmetric}} P(S)\log|\mathrm{Aut}(S)| \quad (\because |\mathrm{Aut}(S)| = 1 \text{ for all asymmetric } S) \\
&\le \sum_{S\in\mathcal{S}(n,p)\text{ is symmetric}} P(S)\cdot n\log n \quad (\because |\mathrm{Aut}(S)| \le n! \le n^n) \\
&= O\left(\frac{\log n}{n^{w-1}}\right) \quad \text{for any positive constant } w > 1 \quad \text{(by Lemma 2).}
\end{aligned}$$

To prove part (ii), we define the *typical set* T_ϵ^n as the set of structures S on n vertices having the following two properties: (a) S is asymmetric; (b) for $G \cong S$,

$$2^{-\binom{n}{2}(h+\epsilon)} \le P(G) \le 2^{-\binom{n}{2}(h-\epsilon)}.$$

Let T_1 and T_2 be the sets of structures satisfying the properties (a) and (b), respectively. Then, $T_\epsilon^n = T_1 \cap T_2$. By the asymmetry of $\mathcal{G}(n,p)$, we know that $P(T_1) > 1-\epsilon$ for large n. As explained above, a labeled graph G can be viewed as a binary sequence of length $\binom{n}{2}$. Thus, by the property (b) and the AEP for binary sequences, we also know that $P(T_2) > 1-\epsilon$ for large n. Thus, $P(T_\epsilon^n) = 1 - P(\overline{T_1} \cup \overline{T_2}) > 1 - 2\epsilon$. Now let us compute $P(S)$ for S in T_ϵ^n. By the property (a), $P(S) = n!P(G)$ for any $G \cong S$. By this and the property (b), we can see that any structure S in T_ϵ^n satisfies the condition in (3). This completes the proof. ∎

Remark 1. The structural entropy can be equivalently written as

$$H_{\mathcal{S}} = \binom{n}{2}h - n\log n + n\log e - \frac{1}{2}\log n - \frac{1}{2}\log(2\pi) + o(1) \tag{4}$$

by Stirling's approximation, $n! \approx \sqrt{2\pi n}\,\left(\frac{n}{e}\right)^n$.

Remark 2. Roughly speaking, Theorem 1(ii) means that the probability of a typical graph structure is $P(S) \sim 2^{-\binom{n}{2}h+\log n!}$.

By Shannon's source coding theorem, the structural entropy computed in Theorem 1 is a fundamental lower bound on the lossless compression of structures from $\mathcal{S}(n,p)$. In the next section, we design an asymptotically optimal compression algorithm matching the first two leading terms as in (4) of the structural entropy with high probability.

As already observed in the introduction, there are other measures of information content of a graph. For example, consider partitioning vertices of a graph G into subsets, $O_i(G)$, such vertices belonging to the same subset have the same long term connectivity. For example, in Figure 2(a) we find that $O_1 = \{v_1, v_4\}$ and $O_2 = \{v_2, v_3\}$. These subsets O_i's turn out to be the so-called *orbits* of the underlying graph automorphism [14]. Clearly, all graphs G of the same structure $S \in \mathcal{S}$ have the same orbits. Assigning some probability measure on the set of orbits, one can define another information metric that can be called the *topological entropy*, H_T [29, 39]. For a given structure S we define the probability on an orbit $O_i(S)$ to be $|O_i(S)|/n$. Then the topological entropy is defined as

$$H_T = -\sum_{S\in\mathcal{S}} P(S) \sum_i \frac{|O_i(S)|}{n} \log \frac{|O_i(S)|}{n},$$

where the sum is over all structures $S \in \mathcal{S}$ and over all enumeration of orbits.

Let us again consider the Erdős-Rényi model for graph generation. By Lemma 2 we conclude that all orbits are singletons with high probability. This leads to the following corollary.

Corollary 1 *Assume graphs are generated according to the Erdős-Rényi process $\mathcal{G}(n,p)$. For all p satisfying $\frac{\ln n}{n} \ll p$ and $1-p \gg \frac{\ln n}{n}$, the topological entropy is*

$$H_T = \log n - \Theta\left(\frac{\log n}{n^\alpha}\right)$$

for $\alpha > 0$.

3.2 Compression Algorithm

Our algorithm is a compression scheme for unlabeled graphs. In other words, given a labeled graph G, it compresses G into a code, from which one can construct a graph S that is isomorphic to G. The algorithm consists of two stages. First it encodes G into two binary sequences and then compresses them using an arithmetic encoder. It achieves the structural entropy up to the first two leading terms shown in (4). In Section 4, we prove our main findings that we summarize below.

Theorem 2 *Let $L(S)$ be the length of the code generated by our algorithm for all graphs G from $\mathcal{G}(n,p)$ that are isomorphic to a structure S. The following holds:*
(i) *For large n,*

$$\mathbf{E}[L(S)] \le \binom{n}{2}h - n\log n + (c + \Phi(\log n))\,n + o(n),$$

where $h := h(p)$, c is an explicitly computable constant, and $\Phi(\log n)$ is a fluctuating function with a small amplitude.
(ii) *Furthermore, for any $\epsilon > 0$,*

$$P\left(L(S) - \mathbf{E}[L(S)] \leq \epsilon n \log n\right) \geq 1 - o(1).$$

(iii) *Finally, our algorithm runs in $O(n+e)$ on average, where e is the number of edges.*

We next describe the algorithm in some details, starting with a general framework and then proposing some useful data structures that allow us to reduce the time complexity to $O(n+e)$.

3.2.1 General Framework

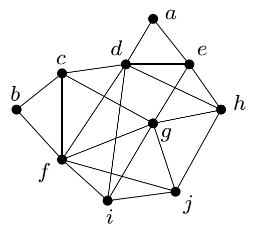
First we need some definitions and notations. An *ordered partition* of a set X is a sequence of nonempty subsets of X such that every element in X is in exactly one of these subsets. For example, one ordered partition of $\{a,b,c,d,e\}$ is $\{a,b\},\{e\},\{c,d\}$ that is denoted by $ab/e/cd$. It is equivalent to $ba/e/dc$, but distinct from $e/ab/cd$. Given an ordered partition P of a set X, we also define an order of the elements of X as follows: $a < b$ in P if the subset containing a precedes the subset containing b in P. For example, $a < c$ and $e < c$ in $P = ab/e/cd$, but $e \not< a$. An ordered partition P_1 of a set X is called *finer* than an ordered partition P_2 of X if the following two conditions hold: (1) every element (i.e., subset of X) of P_1 is a subset of some element of P_2, and (2) for all $a, b \in X$, $a < b$ in P_1 if $a < b$ in P_2. For example, both $a/b/e/cd$ and $ab/e/d/c$ are finer than $ab/e/cd$. Finally, a subtraction of an element from an ordered partition gives us another ordered partition (e.g., for $P = ab/e/cd$ we find that $P - c$ and $P - e$ are $ab/e/d$ and ab/cd, respectively).

The first stage of our algorithm consists of n steps, updating in each step an ordered partition P of a subset of $V(G)$. Let P_i be the partition after the i-th step. At the beginning, $P_0 = V(G)$. In the i-th step, any vertex v is selected to be removed from the first subset in P_{i-1}. Then, for each subset U in $P_{i-1} - v$ (in its order), we encode the *number of neighbors* of v in U using $\lceil \log(|U|+1) \rceil$ bits. After that, $P_{i-1} - v$ becomes a finer partition P_i such that for each subset U in $P_{i-1} - v$, U is divided into two smaller subsets U_1 and U_2, and U_1 precedes U_2 in P_i where U_1 is the set of all neighbors of v in U and U_2 is the set of all non-neighbors of v in U. These steps are repeated until P becomes empty.

While the algorithm is running, the binary encodings of the number of neighbors are concatenated in the order they are generated. During the course of the algorithm, we separately maintain two types of encodings – those of length more than one bits (i.e., for subsets $|U| > 1$) and those of length exactly one bit (i.e., for subsets $|U| = 1$). The former type of encodings are appended to a binary sequence B_1. Similarly, the latter type of encodings form a binary sequence B_2.

Example: Figure 3 shows the progress of our algorithm step by step. Here k denotes the step number, and v denotes the chosen vertex in each step. All encodings whose length is larger than one (denoted by *italic* font) are appended to B_1. The other encodings (those of length one) form B_2. After ten steps, B_1 and B_2 are *0100110100001110101* and 1001011000000101, respectively. ■

In the second stage, B_1 and B_2 are compressed to $\hat{B}_1$ and $\hat{B}_2$ by a binary arithmetic encoder [8]. Finally, the encoding of G consists of n, $\hat{B}_1$, and $\hat{B}_2$.



k	v	$P_{k-1} - v$	encoding	P_k
0				$abcdefghij$
1	i	$abcdefghj$	*0100*	$dfgj/abceh$
2	f	$dgj/abceh$	*11, 010*	$dgj/bc/aeh$
3	d	$gj/bc/aeh$	*00, 01, 11*	$gj/c/b/aeh$
4	j	$g/c/b/aeh$	1, 0, 0, *01*	$g/c/b/h/ae$
5	g	$c/b/h/ae$	1, 0, 1, *01*	$c/b/h/e/a$
6	c	$b/h/e/a$	1, 0, 0, 0	$b/h/e/a$
7	b	$h/e/a$	0, 0, 0	$h/e/a$
8	h	e/a	1, 0	e/a
9	e	a	1	a
10	a			

Figure 3: An example for our encoding algorithm, given the graph on the left.

We next describe our decoding algorithm constructing from n, $\hat{B}_1$, and $\hat{B}_2$ a graph isomorphic to the original graph. First we restore B_1 and B_2 by decompressing $\hat{B}_1$ and $\hat{B}_2$. Then, we create a graph G having n vertices and no edges. The general framework of our decoding algorithm is very similar to that of our encoding algorithm. Again, one ordered partition P of a subset of $V(G)$ is maintained. Let P_i be the ordered partition after the i-th step. At the beginning, $P_0 = V(G)$. In the i-th step, we remove any vertex v from the first subset in P_{i-1}. Then, for each subset U in $P_{i-1} - v$ (in its order), we extract the first $\ell = \lceil \log(|U| + 1) \rceil$ bits from either B_1 (if $|U| > 1$) or B_2 (if $|U| = 1$), and we select any ℓ vertices in U and make an edge between v and each of those ℓ vertices. After that, $P_{i-1} - v$ becomes a finer partition P_i in the same way as our encoding algorithm. These steps are repeated until P becomes empty.

Example: Let us reconstruct a graph from the encoding in the previous example. After decompressing we have n=10, B_1=*0100110100001110101*, and B_2=1001011000000101. We start with a graph of 10 isolated vertices, and proceed as described above. Figure 4 shows the details. Again, k denotes the step number, and v denotes the chosen vertex (here we just select the first vertex.) The last column shows the edges created in the k-th step. The extracted bits from B_1 are denoted by *italic* font. On the right is shown the reconstructed graph, which is isomorphic to the original graph. ■

k	v	$P_{k-1} - v$	extracted bits	P_k	created edges
0				$abcdefghij$	
1	a	$bcdefghij$	*0100*	$bcde/fghij$	$\{a,b\}, \{a,c\}, \{a,d\}, \{a,e\}$
2	b	$cde/fghij$	*11, 010*	$cde/fg/hij$	$\{b,c\}, \{b,d\}, \{b,e\}, \{b,f\}, \{b,g\}$
3	c	$de/fg/hij$	*00, 01, 11*	$de/f/g/hij$	$\{c,f\}, \{c,h\}, \{c,i\}, \{c,j\}$
4	d	$e/f/g/hij$	1, 0, 0, *01*	$e/f/g/h/ij$	$\{d,e\}, \{d,h\}$
5	e	$f/g/h/ij$	1, 0, 1, *01*	$f/g/h/i/j$	$\{e,f\}, \{e,h\}, \{e,i\}$
6	f	$g/h/i/j$	1, 0, 0, 0	$g/h/i/j$	$\{f,g\}$
7	g	$h/i/j$	0, 0, 0	$h/i/j$	
8	h	i/j	1, 0	i/j	$\{h,i\}$
9	i	j	1	j	$\{i,j\}$
10	j				

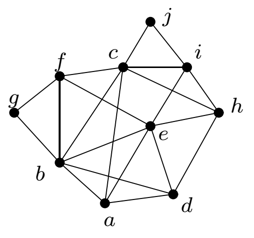


Figure 4: An example for our decoding algorithm, given n=10, B_1=*0100110100001110101* and B_2=1001011000000101 (the reconstructed graph is shown on the right.)

In a naive implementation of the general framework of our encoding algorithm, the time complexity is $O(n^2)$ as follows. In each step of the first stage, we need to count the number of neighbors in each disjoint subset in P and split it into two smaller subsets. This can be done in $O(n)$ time by scanning all remaining vertices in P. Thus the first stage takes $O(n^2)$

time in total. In the second stage, a linear-time arithmetic encoder takes $O(n^2)$ time since the length of B_2 is $\Theta(n^2)$, which will be proved in Section 4.

To reduce the time complexity to $O(n+e)$, we shall use the following three novel techniques described in details below. First, we use efficient data structures for maintaining the partition P and encoding the number of neighbors in each subset. Second, in the arithmetic encoding, we process the intermediate sequence B_2 not in bitwise manner, but instead we process a run of consecutive zeroes in one step. Third, when outputting the code in the arithmetic encoder, we use a greedy outputting method proposed in [20].

3.2.2 Data structures

To describe our data structures, we define the *position* of a vertex v in a partition P as the number of vertices on the right side of v in P. Similarly, we define the *rank* of a subset U and all vertices $v \in U$ as the number of vertices on the right side of U in P (i.e., the position of the rightmost vertex in U).

The partition P of a subset of $V(G)$ is maintained by the following five arrays, each of which is of size n. Arrays $pos[v]$ and $rank[v]$ store the position and the rank of a vertex v in P, respectively. An array $vertex[i]$ stores the vertex at position i (i.e., $pos[vertex[i]] = i$). An array $size[r]$ stores the size of a subset whose rank is r. Lastly, for r such that $size[r] > 1$, an array $next[r]$ stores the largest rank r' such that $r > r'$ and $size[r'] > 1$. We also have a variable *head* containing the largest rank r such that $size[r] > 1$. These arrays are updated while P becomes smaller and finer in each step. For instance, when $P = P_3 = gj/c/b/aeh$ in our previous example, the arrays are as follows.

	j	i	h	g	f	e	d	c	b	a
pos	5	-	0	6	-	1	-	4	3	2
rank	5	-	0	5	-	0	-	4	3	0

	9	8	7	6	5	4	3	2	1	0
vertex	-	-	-	g	j	c	b	a	e	h
size	0	0	0	0	2	1	1	0	0	3
next	-	-	-	-	0	-	-	-	-	-

We observe the following properties: (1) The vertices with the same rank are in the same subset in P; (2) The division of a subset U does not affect the ranks of vertices outside U (in fact, it affects only the rank of vertices in U that are neighbors of the chosen vertex); (3) Once the size of a subset becomes one, its rank is the same as its position and does not change until the end; (4) Using *head* and *next*, one can traverse only the subsets whose size is larger than one.

3.2.3 Algorithm

Now we describe our algorithm in more detail. The first stage consists of n steps. Let P_i be the ordered partition after the i-th step, which is maintained implicitly by the arrays described above. Here we assume that the input graph is given as an adjacency list and $N(v)$ denotes the list of neighbors of vertex v. We also have a temporary array $\mathcal{B}$ of size n, in each element of which we have a stack. An array *count* is used for counting the number of neighbors. In the i-th step, the algorithm works as follows:

1. Remove the leftmost vertex v from P_{i-1} and update arrays accordingly.
2. While traversing $N(v)$, for each neighbor u that is still in P_{i-1} (let $r = rank[u]$),
    - 2.1. If $size[r] > 1$, increase $count[r]$ by one.

2.2. If $size[r] = 1$, mark its position by pushing the step number i to the stack in $\mathcal{B}[r]$.

3. While traversing subsets U such that $|U| > 1$ using *head* and *next* (let r be the rank of U),
   3.1. Encode the number of neighbors in U (stored in $count[r]$) using $\lceil \log(size[r] + 1) \rceil$ bits, and output to B_1.
   3.2. Mark the position of U (i.e., the positions of both ends of U) by pushing $-i$ to both stacks in $\mathcal{B}[r]$ and $\mathcal{B}[r + size[r] - 1]$.
   3.3. Update *size* and *next* arrays accordingly reflecting the division of U.
4. While traversing $N(v)$, for each neighbor u that is still in P_{i-1},
   4.1. Update the rank of u.
   4.2. Move u to the correct position by updating *pos* and *vertex* (i.e., swap u and the vertex at the position which u moves to.)

After repeating the above steps until P becomes empty, we extract B_2 from $\mathcal{B}$ in a form of run length codes, that is, a sequence of the lengths of the runs of zeroes between each two consecutive '1's (including both ends). For example, $B_2 = 1001011000000101$ is encoded as $0, 2, 1, 0, 6, 1, 0$. The time complexity of the construction of B_2 is analyzed in the following lemma, which will be used later to analyze the overall time complexity of our algorithm.

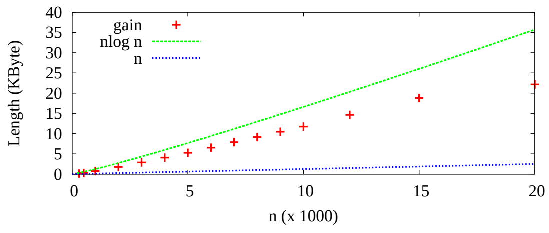
Lemma 3 *The sequence B_2 can be constructed by scanning $\mathcal{B}$ once, and it takes $O(n + \ell)$ time, where ℓ is the total number of elements inserted in $\mathcal{B}$.*

Proof: In the i-th step, there are $n - i$ vertices in P, and their positions are from 0 to $n - i - 1$. In procedure 3.2, the position of each subset of size larger than one in P is marked. Thus, one can infer the number of subsets of size one (i.e., singleton sets) in P and the positions of those subsets. In procedure 2.2, the position of each singleton set containing a neighbor is marked. Each of these marked positions contributes a bit '1' and each of the rest contributes a bit '0'. Thus the concatenation c_i of the bits in decreasing order of position is the contribution to B_2 in the i-th step. Therefore, B_2 is nothing but $c_1 c_2 \cdots c_{n-1}$. Clearly, for each c_i, the number of zeroes between each two consecutive '1's can be computed in one scan of $\mathcal{B}$. This can be done for all i's in parallel. After that, the concatenation of c_i's takes $O(n)$ time. ∎

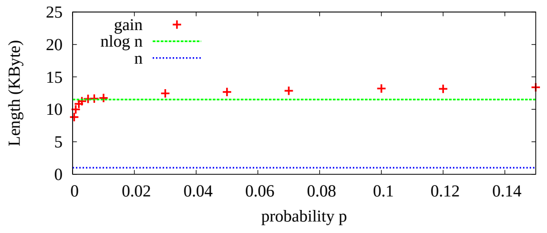
In the second stage, both B_1 and B_2 are compressed by a binary arithmetic encoder, but B_2 is compressed by a modified arithmetic encoder using the greedy outputting method as described in [20]. We first briefly describe a general (non-adaptive) binary arithmetic encoder and then describe our modified arithmetic encoder. Given a probability p for a bit '1', the encoder starts with an initial interval $[0, N)$ where N is a large positive integer. For each bit, it first calculates the new interval from the current interval. Then, it outputs code bits from the newly calculated interval so that its length is greater than a predefined threshold. If we use this encoder, the complexity would be $O(n^2)$ since the length of B_2 is $\Theta(n^2)$ in bits. Thus, in the modified encoder, we process a run of zeroes in one step. When we extract B_2 from $\mathcal{B}$, we estimate the probability p of having '1' in B_2 and also precompute in a table the probability of a run of k zeroes, which is $(1-p)^k$ for $k = 1, 2, \cdots$. We recall that B_2 stores lengths of runs of zeroes. When the encoder gets a number from B_2, it calculates the new

Table 1: The average code length and running time for real-world networks.

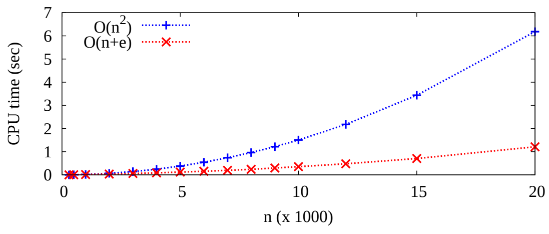
			Code length (bits)				CPU time (secs)	
Networks	# of nodes	# of edges	our algo.	adj. mat. $\binom{n}{2}$	adj. list $e\lceil\log n\rceil$	arithmetic coding	$O(n+e)$ algo.	$O(n^2)$ algo.
US Airports	332	2,126	8,108	54,946	19,134	12,947	<0.01	<0.01
Protein interaction(Yeast)	2,329	6,646	46,853	2,785,980	79,752	67,063	0.11	0.12
Collaboration(Geometry)	6,167	21,535	113,684	19,012,861	279,955	241,549	0.47	0.68
Collaboration(Erdős)	6,934	11,857	60,263	24,043,645	154,141	147,121	1.02	1.08
Genetic interaction(Human)	8,595	26,066	221,226	37,018,710	364,924	310,459	1.22	1.54
Internet(AS level)	25,881	52,407	301,463	334,900,140	786,105	737,851	12.97	13.81



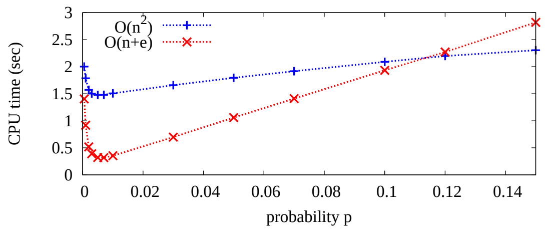
(a) gain in code length against arithmetic coding (p=0.01)



(b) gain in code length against arithmetic coding (n=10000)



(c) running time (p=0.01)



(d) running time (n=10000)

Figure 5: The average gain in code length and running time for the Erdős-Rényi random graphs.

interval for a run of zeroes in constant time by looking up this precomputed table. After this, it outputs code bits in constant time using the greedy outputting method in [20], and then it processes a bit '1' in its usual way. Here we need one restriction on k since for very large k the probability $(1-p)^k$ becomes too small to represent the new interval precisely. Thus, we set $k_{max} = \lceil 1/p \rceil$, and if $k > k_{max}$, then we process only the first k_{max} zeroes in every step until all k zeroes are exhausted.

3.3 Experimental Results

To test our algorithm, we applied it to the Erdős-Rényi random graphs and real-world networks including biological, social, and technological networks. Table 1 summarizes the results for the real-world networks. For comparison, we list the lengths of three other encodings of graphs, namely, the usual implementations of adjacency matrix of $\binom{n}{2}$ bits, and adjacency list of *at least* $e\lceil\log n\rceil$ bits (normally, $2e\lceil\log n\rceil$ bits) where e is the number of edges. Finally, we applied an arithmetic encoder to the adjacency matrix, which can achieve $\binom{n}{2}h(p)$ bits. For many real-world networks, our algorithm achieves twice better compression than the standard arithmetic encoder. For comparison, we also implemented $O(n^2)$-time algorithm [7]. For all of our real-world test data, our $O(n+e)$-time algorithm is faster than $O(n^2)$-time algorithm. We measured CPU time on a machine equipped with Pentium D 3.0GHz processor and 2GB of RAM, running Linux. All the numbers are averages over 100 measurements.

Figure 5 shows the results for $\mathcal{G}(n,p)$ graphs. In (a) and (b), we plot the gain of our encoding against arithmetic encoding, that is, the difference between two encodings. We plot it for a fixed p in (a) and for a fixed n in (b). The plots confirm our analysis that the gain is asymptotically close to $n \log n$. In (c) and (d), we plot the CPU time consumed by $O(n+e)$-time and $O(n^2)$-time algorithms, and it shows that our $O(n+e)$-time algorithm is faster unless the graph is too dense.

Let us make some final observations. Our results predict that for structures from $\mathcal{S}(n,p)$ one can achieve compression up to

$$\binom{n}{2} h(p) - n \log n + O(n)$$

bits which should be compared to $\binom{n}{2} h(p)$ bits, if conventional algorithms are used (i.e., arithmetic encoder to the adjacency matrix). The redundancy, $n \log n$ of our compression scheme is confirmed for randomly generated graphs from $\mathcal{G}(n,p)$. For many real-world graphs, however, our algorithm achieves more than twice better compression when compared to standard arithmetic encoder. While these graphs are not randomly generated according to $\mathcal{G}(n,p)$ (rather by a *power-law* distribution), we believe their compression is a consequence of small p.

Indeed, consider even in our $\mathcal{G}(n,p)$ model the behavior of the structural entropy $H_{\mathcal{S}}$ when $p \to 0$ satisfying the conditions of Theorem 1. Let then $p \sim \omega(n)(\log n/n)$ for slowly growing $\omega(n) \to \infty$ as $n \to \infty$. In this case

$$h(p) \sim \omega(n) \frac{\log^2 n}{n},$$

and therefore the structural entropy becomes

$$H_{\mathcal{S}} \sim \frac{1}{2}(n-1)\omega(n) \log^2 n - n \log n + O(n).$$

Clearly, the second leading term $n \log n$ plays a significant role in the compression of such graphs. This may explain why our encoding is much better than arithmetic coding for real-world networks that are usually sparse graphs.

4 Analysis

In this section, we analyze the compression performance and time complexity of our algorithm, proving Theorem 2. To accomplish it we apply a variety of combinatorial and analytic techniques such as generating functions, Mellin transform, poissonization, and combinatorics.

We start with a description of two binary trees that better capture the progress of our algorithm. Given a graph G on n vertices, the binary tree T_n is built as follows. At the beginning, the root node contains all n graph vertices, $V(G)$, that one can also visualize as n balls. Then a graph vertex (ball) v is removed from the root node. The other $n-1$ graph vertices move down to the left or right depending whether they are adjacent vertices in G to v or not; adjacent vertices go to the left child node and the others go to the right child node. We create a new child node in T_n if there is at least one graph vertex in that node. At this point, the tree is of height 1 with $n-1$ vertices in the nodes at level 1. By induction, in the i-th step, we remove one graph vertex (ball) v from the (level-wise) leftmost node at level $i-1$. The other graph vertices at level $i-1$ move down to the left or right depending whether they are adjacent to v or not. We repeat these steps until all graph vertices are removed (i.e., after n steps).

For our example graph in Figure 3, the construction of the tree T_n and the progress of the algorithm are presented in Figure 6. The removed graph vertices are shown on the left. At each level, the subsets of graph vertices (after removing a vertex from the leftmost node) are shown next to the nodes. We observe that the subsets at each level (from left to right) are the same as the subsets in each step of our algorithm in Figure 3 since we removed the same vertices.

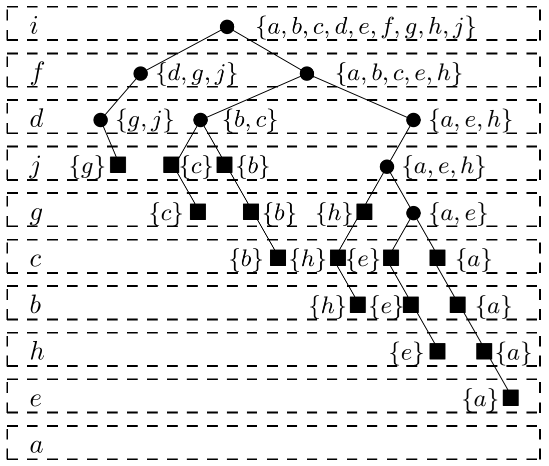


Figure 6: A binary tree T_n with square-shaped nodes containing exactly one ball.

Let N_x denote the number of graph vertices that pass through node x in T_n (excluding the graph vertex removed at x, if any). In Figure 6, for example, N_x is the number of graph vertices shown next to the node x. Our algorithm needs to encode, for each node x in T_n, the number of neighbors (of the removed graph vertex) among N_x vertices. This requires $\lceil \log(N_x+1) \rceil$ bits. Let $L(B_1)$ and $L(B_2)$ be the lengths of sequences B_1 and B_2, respectively. Then, by the construction

$$L(B_1) = \sum_{x \in T_n \text{ and } N_x > 1} \lceil \log(N_x+1) \rceil,$$

$$\text{and} \quad L(B_2) = \sum_{x \in T_n \text{ and } N_x = 1} \lceil \log(N_x+1) \rceil = \sum_{x \in T_n \text{ and } N_x = 1} 1.$$

In Figure 6, the summations for $L(B_1)$ and $L(B_2)$ are over all circle-shaped nodes and over all square-shaped nodes, respectively. Here we can observe an important property of B_2 presented next.

Lemma 4 *Given a graph from $\mathcal{G}(n,p)$, the sequence B_2 constructed by our algorithm is probabilistically equivalent to a binary sequence generated by a memoryless source(p) with p being the probability of generating a '1'.*

Proof: Consider any one bit $b \in B_2$. It represents the number of neighbors of a vertex u in a subset, which contains only one vertex, say v. Then the probability that $b =$'1' is the same as the probability that u and v are connected, which is p. Let us consider any two bits b_1 and b_2. Assume that b_i corresponds to vertices u_i and v_i (i.e., b_i corresponds to the potential edge between u_i and v_i.) These two potential edges are chosen independently. This shows the memoryless property. ∎

To set up precise recurrence relations for our analysis, we need to define a random binary tree $T_{n,d}$ for integers $n \geq 0$ and $d \geq 0$, which is generated similarly to T_n as follows. If $n = 0$, then it is just an empty tree. For $n > 0$, we create a root node, in which we put n balls. In each step, all balls independently move down to the left (with probability p) or right (with probability $1 - p$). We create a new node if there is at least one ball in that node. Thus, after the i-th step, the balls will be at level i. If the balls are at level d or greater, then we remove one ball from the leftmost node before the balls move down to the next level. These steps are repeated until all balls are removed (i.e., after $n + d$ steps). We observe that, if T_n is generated by a graph from $\mathcal{G}(n, p)$, T_n is nothing but the random binary tree $T_{n,0}$. Thus, by analyzing $T_{n,0}$, we can compute both $L(B_1)$ and $L(B_2)$.

4.1 Proof of Theorem 2(i): Average Performance

In this section, we prove part (i) of our main result, that is, we derive the average length of the compressed string representing graphical structure.

Let us first estimate $L(B_1)$. As before, N_x denotes the number of balls that pass through node x (excluding the ball removed at x if any). Let

$$A_{n,d} = \sum_{x \in T_{n,d} \text{ and } N_x > 1} \lceil \log(N_x + 1) \rceil,$$

and $a_{n,d} = \mathbf{E}[A_{n,d}]$. Then $\mathbf{E}[L(B_1)] = a_{n,0}$. Clearly, $a_{0,d} = a_{1,d} = 0$ and $a_{2,0} = 0$. For $n \geq 2$ and $d = 0$, we observe that

$$a_{n+1,0} = \lceil \log (n+1) \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (a_{k,0} + a_{n-k,k}). \tag{5}$$

This follows from the fact that starting with $n + 1$ balls in the root node, and removing one ball we are left with n balls passing through the root node. This contributes $\lceil \log (n + 1) \rceil$. Then, those n balls move down to the left or right subtrees. Let us assume k balls move down to the left subtree (the other $n-k$ balls must move down to the right subtree, and this happens with probability $\binom{n}{k} p^k q^{n-k}$.) At level one, one ball is removed from those k balls in the root of the left subtree. This contributes $a_{k,0}$. There will be no removal among $n - k$ balls in the right subtree until all k balls in the left subtree are removed. This contributes $a_{n-k,k}$. Similarly, for $d > 0$, we can see that

$$a_{n,d} = \lceil \log (n+1) \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (a_{k,d-1} + a_{n-k,k+d-1}). \tag{6}$$

This recurrence is quite complex, but we only need a good upper bound that is presented in the next lemma.

Lemma 5 *For all integers $n \geq 0$ and $d \geq 0$,*

$$a_{n,d} \leq x_n$$

such that x_n satisfies $x_0 = x_1 = 0$ and for $n \geq 2$

$$x_n = \lceil \log (n+1) \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k}). \tag{7}$$

Proof: We use induction on both n and d. Clearly, $a_{n,d} \leq x_n$ for $n = 0$ or 1 ($d \geq 0$). For $n = 2$ and $d = 0$, $a_{2,0} \leq x_2$ since $a_{2,0} = 0$ and $x_2 \geq 2$. For other cases ($n = 2$ and $d > 0$, or $n > 2$), we assume that $a_{i,j} \leq x_i$ holds for $i < n$, and for $i = n$ and $j < d$. Now we want to show that $a_{n,d} \leq x_n$. We divide it into two cases.
(i) When $d = 0$. We observe that

$$a_{n,0} \leq a_{n+1,0} = \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (a_{k,0} + a_{n-k,k}) + q^n a_{n,0} + p^n a_{n,0}.$$

Thus,

$$(1 - p^n - q^n) a_{n,0} \leq \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (a_{k,0} + a_{n-k,k}). \tag{8}$$

Similarly, from (7), we get

$$(1 - p^n - q^n) x_n = \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k}). \tag{9}$$

Therefore,

$$\begin{aligned}
(1 - p^n - q^n) a_{n,0} &\leq \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (a_{k,0} + a_{n-k,k}) \qquad \text{(by (8))} \\
&\leq \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k}) \quad \text{(by induction hypothesis)} \\
&= (1 - p^n - q^n) x_n. \qquad \text{(by (9))}
\end{aligned}$$

(ii) When $d > 0$. By (6) and induction hypothesis,

$$a_{n,d} \leq \lceil \log (n+1) \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k}) = x_n.$$

This completes the proof. ■

The next step involves solving asymptotically recurrence (7). We do it in Section 5 proving the following lemma.

Lemma 6 *Consider the following recurrence for x_n with $x_0 = x_1 = 0$ and for $n \geq 2$*

$$x_n = a_n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k}),$$

where $a_n = \lceil \log (n+1) \rceil$ for $n \geq 2$ and $a_0 = a_1 = 0$. Then:
(i) *If $\log p / \log q$ is irrational, then*

$$x_n = \frac{n}{h} A^*(-1) \log e + o(n), \tag{10}$$

where

$$A^*(-1) = \sum_{b\geq 2} \frac{\lceil \log(b+1) \rceil}{b(b-1)}. \tag{11}$$

(ii) *If* $\log p/\log q = r/d$ *(rational) with* $\gcd(r,d) = 1$*, then*

$$x_n = \frac{n}{h}\left(A^*(-1) + \Phi(\log_p n)\right)\log e + O(n^{1-\eta}) \tag{12}$$

for some $\eta > 0$*, where*

$$\Phi(x) = \sum_{k\neq 0} A^*(-1 + 2k\pi ri/\log p)\exp(2k\pi rxi) \tag{13}$$

is a fluctuating function with a small amplitude.

Finally, the average length of $L(B_1)$ can be derived. We present it in the next theorem.

Theorem 3 *For large* n*,*

$$\mathbf{E}[L(B_1)] \leq \frac{n}{h}\left(\beta + \Phi_1(\log n)\right) + o(n),$$

where $h := h(p)$*,*

$$\beta = \log e \cdot \sum_{b\geq 2} \frac{\lceil \log{(b+1)} \rceil}{b(b-1)} = 3.760\cdots,$$

and $\Phi_1(\log n)$ *is a fluctuating function for* $\log p/\log q$ *rational with small amplitude and asymptotically zero otherwise.*

The next step is to estimate the average length of B_2. Let $S_{n,d}$ be the total number of nodes x in $T_{n,d}$ such that $N_x = 1$, that is,

$$S_{n,d} = \sum_{x\in T_{n,d} \text{ and } N_x=1} 1 = \sum_{x\in T_{n,d} \text{ and } N_x=1} N_x = \sum_{x\in T_{n,d}} N_x - \sum_{x\in T_{n,d} \text{ and } N_x>1} N_x.$$

Let $B_{n,d} = \sum_{x\in T_{n,d}, N_x>1} N_x$. We observe that

$$L(B_2) = S_{n,0} = \sum_{x\in T_{n,0}} N_x - B_{n,0} = \frac{n(n-1)}{2} - B_{n,0}. \tag{14}$$

The last equality follows from the fact that the sum of N_x's for all x at level ℓ in $T_{n,0}$ is equal to $n-1-\ell$.

Let $b_{n,d} = \mathbf{E}[B_{n,d}]$. For our analysis we only need $b_{n,0}$. Clearly, $b_{0,d} = b_{1,d} = 0$ and $b_{2,0} = 0$. For $n \geq 2$, we can find the following recurrence (similarly to $a_{n,d}$):

$$b_{n+1,0} = n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}(b_{k,0} + b_{n-k,k}), \tag{15}$$

$$\text{and} \quad b_{n,d} = n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}(b_{k,d-1} + b_{n-k,k+d-1}) \quad \text{for } d > 0. \tag{16}$$

To prove our main result, we only need a lower bound that is established in the next lemma.

Lemma 7 *For all $n \geq 0$ and $d \geq 0$,*

$$b_{n,d} \geq y_n - \frac{n}{2}$$

such that y_n satisfies $y_0 = 0$ and for $n \geq 0$

$$y_{n+1} = n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (y_k + y_{n-k}). \tag{17}$$

Proof: We prove it by induction on both n and d. Clearly, $b_{n,d} \geq y_n - n/2$ for $n = 0$ or 1 ($d > 0$). For $n = 2$ and $d = 0$, $b_{2,0} \geq y_2 - 2$ since $b_{2,0} = 0$ and $y_2 = 1$. For other cases ($n = 2$ and $d > 0$, or $n > 2$), we assume that $b_{i,j} \geq y_i - \frac{i}{2}$ holds for $i < n$, and for $i = n$ and $j < d$. Now we want to show that $b_{n,d} \geq y_n - \frac{n}{2}$. We divide it into two cases.
(i) When $d = 0$. By (15) and induction hypothesis,

$$\begin{aligned} b_{n,0} &\geq (n-1) + \sum_{k=0}^{n-1} \binom{n-1}{k} p^k q^{n-1-k} (y_k - \frac{k}{2} + y_{n-1-k} - \frac{n-1-k}{2}) \\ &= y_n - \frac{n-1}{2} > y_n - \frac{n}{2}. \quad \text{(by (17))} \end{aligned}$$

(ii) When $d > 0$. By (16) and induction hypothesis,

$$\begin{aligned} b_{n,d} &\geq n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (y_k - \frac{k}{2} + y_{n-k} - \frac{n-k}{2}) \\ &= y_{n+1} - \frac{n}{2} \geq y_n - \frac{n}{2}. \end{aligned}$$

This completes the proof. ■

It is easy to see that y_n represents the expected path length in a *digital search tree* over n strings as discussed in [17, 38]. The authors of [17] proved, among others, that

$$y_n = \frac{n}{h}\left(\log n + \frac{h_2}{h} + \gamma - 1 - \alpha + \Phi_2(\log n)\right) + \frac{1}{h}\left(\log n + \frac{h_2}{2h} - \gamma - \log p - \log q + \alpha\right) + O(1), \tag{18}$$

where $h_2 = p \log^2 p + q \log^2 q$, $\gamma = 0.577\cdots$ is the Euler constant, and

$$\alpha = -\sum_{k=1}^{\infty} \frac{p^{k+1} \log p + q^{k+1} \log q}{1 - p^{k+1} - q^{k+1}}.$$

In the above, $\Phi_2(\log n)$ is a fluctuating function for $\log p / \log q$ rational with small amplitude and zero otherwise.

In summary, by (14), Lemma 7, and the above, we arrive at our next result.

Theorem 4 *For large n,*

$$\mathbf{E}[L(B_2)] \leq \frac{n(n-1)}{2} - \frac{n}{h}\log n + \frac{n}{h}\left(\frac{h}{2} - \frac{h_2}{h} - \gamma + 1 + \alpha - \Phi_2(\log n)\right) - \frac{1}{h}\log n + O(1),$$

with the notations as below (18).

Finally, we compute $\mathbf{E}[L(S)] = \mathbf{E}[L(\hat{B_1}) + L(\hat{B_2})] + O(\log n)$, proving the part (i) of Theorem 2. We observe that the arithmetic encoder can compress a binary sequence of length m on average up to $mh + \frac{1}{2}\log m + O(1)$, where h is the entropy rate of the binary source [9, 41]. Thus, by Theorem 3,

$$\mathbf{E}[L(\hat{B_1})] \leq \frac{h'}{h}(\beta + \Phi_1(\log n))n + o(n),$$

where h, β, and $\Phi_1(\log n)$ are defined in Theorem 3, and h' is the entropy rate of the binary source that B_1 is generated from. Similarly, we can compute $\mathbf{E}[L(\hat{B_2})]$. In this case, however, we know that the entropy rate for B_2 is $h := h(p)$. Thus, by Theorem 4,

$$\mathbf{E}[L(\hat{B_2})] \leq \binom{n}{2}h - n\log n + n\left(\frac{h}{2} - \frac{h_2}{h} - \gamma + 1 + \alpha - \Phi_2(\log n)\right) + O(\log n),$$

where h, h_2, γ, α, and $\Phi_2(\log n)$ are defined above. This completes the part (i) of Theorem 2.

4.2 Proof of Theorem 2(ii): Performance with High Probability

Now we prove part (ii) of Theorem 2, that is, we show that $L(S) - \mathbf{E}[L(S)] \leq \epsilon n \log n$ with high probability. Since $L(S) = L(\hat{B_1}) + L(\hat{B_2})$, we need bounds for $L(\hat{B_1})$ and $L(\hat{B_2})$. We start with $L(\hat{B_1})$. By Markov's inequality,

$$P\left(L(\hat{B_1}) > \epsilon n \log n\right) < \frac{\mathbf{E}[L(\hat{B_1})]}{\epsilon n \log n} = O\left(\frac{1}{\log n}\right), \quad \epsilon > 0. \tag{19}$$

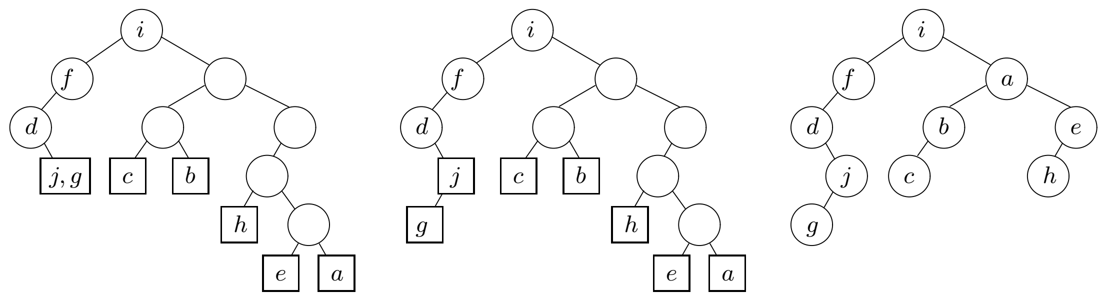
Handling $L(\hat{B_2})$ is more complicated. In [41] it was proved that for a binary sequence X of length ℓ, the code length generated by an arithmetic encoder is at most $-\log P(X) + \frac{1}{2}\log \ell + 3$. In our case, $B_2 = b_1 b_2 \cdots b_{L(B_2)}$ is memoryless, and then

$$L(\hat{B_2}) < -\log P(B_2) + \frac{1}{2}\log L(B_2) + 3 = L(B_2)\cdot\left[-\frac{1}{L(B_2)}\sum_{i=1}^{L(B_2)}\log P(b_i)\right] + \frac{1}{2}\log L(B_2) + 3. \tag{20}$$

Thus we need good bounds for $L(B_2)$ and the sum of $\log P(b_i)$. With respect to $L(B_2)$, recall that $L(B_2) = \binom{n}{2} - B_{n,0}$ where

$$B_{n,0} = \sum_{x \in T_{n,0}, N_x > 1} N_x,$$

and N_x is the number of balls that pass through node x in tree $T_{n,0}$ (excluding the ball removed at x if any). We shall show that $B_{n,0}$ is related to the *path lengths* in slightly modified trees that we denote as $\hat{T}_n$ and $\bar{T}_n$. The tree $\hat{T}_n$ is constructed from $T_{n,0}$ by removing all nodes x with $N_x = 1$ that are not direct children of nodes y with $N_y > 1$. Then, we put back balls into the nodes of $\hat{T}_n$ using the following rules: we put each ball back into the node where it was removed; if such a node does not exist in $\hat{T}_n$, then we put the ball into the first node x with $N_x = 1$ on its path in $T_{n,0}$. To construct $\bar{T}_n$ we observe that there might be some nodes with two balls in $\hat{T}_n$. In such a case, we add a child node and move one ball down to the new node to eliminate all nodes with two balls. Figure 7(a,b) illustrates the construction of



(a) $\hat{T}_n$ (b) $\bar{T}_n$ (c) D_n

Figure 7: An example of binary trees $\hat{T}_n$, $\bar{T}_n$, and D_n, given binary choices for 10 balls $\{i,f,d,j,g,a,b,c,e,h\}$.

$\hat{T}_n$ and $\bar{T}_n$ for the tree T_n (equivalently, $T_{n,0}$) from Figure 6. Notice that in this figure all circle-shaped nodes and the square-shaped nodes – directly connected to these circle-shaped nodes – are the same in both T_n and $\hat{T}_n$.

Let $\ell(\hat{T}_n)$ and $\ell(\bar{T}_n)$ be the path lengths to all *balls* in $\hat{T}_n$ and $\bar{T}_n$, respectively. From the construction it is clear that

$$B_{n,0} = \ell(\hat{T}_n).$$

Now let us compare $\ell(\hat{T}_n)$ and $\ell(\bar{T}_n)$. Whenever we have two balls in a node of $\hat{T}_n$, we move one ball down in $\bar{T}_n$ to a new node. This results in such a path in $\bar{T}_n$ being longer by one than the corresponding path in $\hat{T}_n$. However, this can happen at most $n/2$ times since there are at most $n/2$ nodes with two balls. Thus we find[2]

$$\ell(\hat{T}_n) + n/2 \geq_{st} \ell(\bar{T}_n).$$

To estimate the path length $\ell(\bar{T}_n)$, we introduce another binary tree D_n that is probabilistically equivalent to the *digital search tree* built over n random binary strings. It is constructed as follows. If $n = 0$, then it is just an empty tree. For $n > 0$, we create a root node in which we put n balls. One ball remains in the root node, and rest of balls independently move down to the left or right. We create a new child node if there is at least one ball in that node. We recursively repeat it (i.e., we leave one ball in a node while moving others down.) Figure 7(c) illustrates this construction.

We shall next show that

$$\ell(\bar{T}_n) \geq_{st} \ell(D_n),$$

where $\ell(D_n)$ is the path length to all balls (nodes) in D_n. For this, we consider two actual trees $\bar{t}_n$ and d_n given the same binary choices regarding the action left/right (1/0) for the n balls. We also assume that the input to both trees is the same, that is, balls are inserted in the same order and therefore we always identify the "smallest" ball in input. Whenever a ball remains in a node during the construction of these trees, we assume that the smallest

[2]For two real-valued random variables X and Y, we write $X \geq_{st} Y$ if the value of X is always greater than or equal to that of Y for every event, or equivalently if $P(X > t) \geq P(Y > t)$ for all $t \in (-\infty, \infty)$ [31].

ball is left in the node. Then, in the next lemma we show that the path length in $\bar{t}_n$ is at least the path length in d_n. Thus $\ell(\bar{T}_n) \geq_{st} \ell(D_n)$.

Lemma 8 *Given binary choices for n balls, let $\bar{t}_n$ and d_n be two tree instances of $\bar{T}_n$ and D_n, respectively. Let $u_t \in \bar{t}_n$ and $u_d \in d_n$ be two corresponding nodes in these trees (i.e., nodes that are reached by the same binary choices). We denote by $B(u)$ the set of balls in the subtree rooted at node u. Then, $B(u_t) \supset B(u_d)$ for any $u_t \in \bar{t}_n$ and $u_d \in d_n$.*

Proof: For the root nodes, it is trivial since both sets have the same n balls. Now it is sufficient to show that the statement is true for children if it is true for their parent nodes. Thus let us assume that $B(u_t) \supset B(u_d)$ for $u_t \in \bar{t}_n$ and $u_d \in d_n$. Let s_t and s_d be the smallest ball (in the input ordering) in $B(u_t)$ and $B(u_d)$, respectively. Now we consider two sets of balls S_t and S_d that will move down from u_t and u_d, respectively. Note that $S_d = B(u_d) - s_d$. We shall show that $S_t \supset S_d$ considering two cases: 1) if u_t is not the leftmost node, then $S_t = B(u_t) \supset B(u_d) - s_d = S_d$; 2) if u_t is the leftmost node, then $S_t = B(u_t) - s_t \supset B(u_d) - s_d = S_d$ since either s_t is the same ball as s_d or s_t is not in S_d. Therefore each ball $b \in S_d$ is also in S_t, and b moves down in the same direction for both u_t and u_d. Therefore, the statement is true for both children nodes. ■

Now we are ready to prove a relation between $B_{n,0}$ and the path length in a digital search tree shown in the following lemma.

Lemma 9 *Let $Y_n := \ell(D_n)$ be the path length in a digital search tree. Then,*

$$B_{n,0} + \frac{n}{2} \geq_{st} Y_n.$$

Proof: Given binary choices for n balls, let us consider tree instances $\hat{t}_n$, $\bar{t}_n$, and d_n. As we observed, $\ell(\hat{t}_n) + \frac{n}{2} \geq \ell(\bar{t}_n) \geq \ell(d_n)$. Therefore, $\ell(\hat{T}_n) + \frac{n}{2} \geq_{st} \ell(D_n)$. We know that $\ell(\hat{T}_n)$ and $\ell(D_n)$ are equivalent to $B_{n,0}$ and Y_n, respectively. This completes the proof. ■

Finally, we establish the following two lemmas.

Lemma 10 *For any $\epsilon > 0$,*

$$P\left(L(B_2) \leq \binom{n}{2} - y_n + \epsilon y_n\right) \geq 1 - o(1),$$

where y_n is defined in Lemma 7.

Proof: We observe that $y_n = \mathbf{E}[Y_n]$, where Y_n is the path length in a digital search tree. Let us compute the probability $P_n = P\left(L(B_2) > \binom{n}{2} - y_n + \epsilon y_n\right)$ for large n. We shall prove that $P_n \to 0$. We have

$$\begin{aligned}
P_n &= P\left(B_{n,0} < (1-\epsilon)y_n\right) \qquad \text{(by (14), that is, } L(B_2) = \tbinom{n}{2} - B_{n,0}) \\
&\leq P\left(Y_n - \frac{n}{2} < (1-\epsilon)y_n\right) \qquad \text{(by Lemma 9)} \\
&= P\left(\frac{Y_n - y_n}{\sqrt{\mathbf{Var}\ Y_n}} < \frac{-\epsilon y_n + n/2}{\sqrt{\mathbf{Var}\ Y_n}}\right) \\
&\leq P\left(\left|\frac{Y_n - y_n}{\sqrt{\mathbf{Var}\ Y_n}}\right| > \left|\frac{-\epsilon y_n + n/2}{\sqrt{\mathbf{Var}\ Y_n}}\right|\right) \qquad (\because \tfrac{-\epsilon y_n + n/2}{\sqrt{\mathbf{Var}\ Y_n}} < 0 \text{ for large } n) \\
&< A\mu^k \qquad \text{(by Theorem 1A of [17])}
\end{aligned}$$

for positive constants A and $\mu < 1$, where $k = \left|\frac{-\epsilon y_n + n/2}{\sqrt{\mathbf{Var}\ Y_n}}\right| = \Theta(\sqrt{n \log n})$ as proved in Theorem 1A of [17]. Thus, P_n becomes exponentially small as $n \to \infty$. ■

In view of (20) and Lemma 10, we need to find a bound for $\sum_{i=1}^{L(B_2)} \log P(b_i)$ which we present next.

Lemma 11 *For any $\epsilon > 0$,*

$$P\left(-\frac{1}{L(B_2)} \sum_{i=1}^{L(B_2)} \log P(b_i) \le h + \epsilon \frac{\log n}{n}\right) \ge 1 - o(1),$$

where $h := h(p)$.

Proof: Let $F_m(X_1, \cdots, X_m) = -\log P(X_1, \cdots, X_m) - mh$, where X_i's are binary independent random variables with p being the probability of '1' and $q = 1 - p$. Denoting by $\hat{X}_i$ an independent copy of X_i (with the same distribution as X_i), we have

$$|F_m(X_1, \cdots, X_i, \cdots, X_m) - F_m(X_1, \cdots, \hat{X}_i, \cdots, X_m)| \le |\log P(X_i) - \log P(\hat{X}_i)| \le c,$$

where $c = \max\{\log p/q, \log q/p\}$. Thus, by Azuma's inequality [38]

$$P(-\log P(X_1, \cdots, X_m) - mh \ge \epsilon' n \log n) \le \exp\left(-\frac{\epsilon'^2 n^2 \log^2 n}{2mc^2}\right) = o(1)$$

provided that $m = O(n^2)$. Since $L(B_2) = O(n^2)$, this completes the proof. ■

By the above two lemmas, after some algebra we conclude that, with probability $1 - o(1)$,

$$L(\hat{B}_2) < \binom{n}{2} h - n \log n + \epsilon n \log n.$$

This and (19) complete the part (ii) of Theorem 2.

4.3 Proof of Theorem 2(iii): Time Complexity

In this section, we prove part (iii) of Theorem 2, that is, we show that the time complexity of our algorithm in Section 3.2.3 is $O(n+e)$ on average. Let us first analyze the first stage, which consists of n steps. Clearly, the procedure 1 takes constant time in each step, and thus it takes $O(n)$ time in total. The procedures 2 and 4 take $O(|N(v)|)$ time in each step since each of operations inside the loop takes constant time. Thus, they take $\sum_{v \in V(G)} O(|N(v)|) = O(e)$ time in total. In the i-th step, the procedure 3 takes $O(s_i)$ time, where s_i is the number of subsets in $P_{i-1} - v$ whose size is larger than one. Thus, in total, it takes $O(s)$ time where $s = \sum_{i=1}^{n} s_i$, which is the total number of nodes x in $T_{n,0}$ with $N_x > 1$. In Figure 6, for example, s is the number of circle-shaped nodes in T_n. By the same analysis as in Section 5 (in this case, $a_n = 1$ in (21)), we can prove that the expected value of s is at most $O(n)$.

Finally, by Lemma 3, the construction of B_2 from $\mathcal{B}$ takes $O(n + \ell)$ time where ℓ is the number of elements inserted in $\mathcal{B}$. We can see that $\ell = O(e + s)$ as follows. The number of elements inserted in procedure 2.2 is bounded by e since every insertion corresponds to a distinct edge. Clearly, the number of elements inserted in procedure 3.2 is bounded by $O(s)$. Therefore, the first stage takes $O(n+e)$ time on average.

In the second stage, B_1 and B_2 are compressed by an arithmetic encoder. Clearly, B_1 can be compressed in $O(n)$ time since the length of B_1 is $O(n)$. The number of elements in the run length form of B_2 is at most $e+1$. Thus, the time complexity of the compression of B_2 would be $O(e)$ except that there could be long runs of zeroes, which are compressed in multiple steps. Let n_0 and n_1 be the number of '0's and '1's in B_2, respectively. Thus, $p = n_1/(n_0+n_1)$. The number of additional steps to process $k_{max} = \lceil 1/p \rceil$ zeroes is bounded by

$$\frac{n_0}{\lceil 1/p \rceil} \leq \frac{n_0}{1/p} = \frac{n_0 n_1}{n_0+n_1} \leq n_1 \leq e.$$

Therefore, the second stage takes $O(n+e)$ time. This completes the proof.

5 Proof of Lemma 6: Analysis of x_n

In this section, we prove Lemma 6. We shall analyze asymptotically x_n satisfying $x_0 = x_1 = 0$ and for $n \geq 2$

$$x_n = a_n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k}), \tag{21}$$

where $a_n = \lceil \log(n+1) \rceil$ for $n \geq 2$ and $a_0 = a_1 = 0$.

Define the exponential generating function (EGF) of x_n as

$$x(z) = \sum_{n=0}^{\infty} x_n \frac{z^n}{n!}$$

for complex z. Then, from (21), for $n \geq 2$ we have

$$\begin{aligned} \frac{x_n}{n!} z^n &= \frac{a_n}{n!} z^n + \sum_{k=0}^{n} \frac{1}{k!} (zp)^k \frac{1}{(n-k)!} (zq)^{n-k} (x_k + x_{n-k}) \\ &= \frac{a_n}{n!} z^n + \sum_{k=0}^{n} \frac{x_k}{k!} (zp)^k \frac{1}{(n-k)!} (zq)^{n-k} + \sum_{k=0}^{n} \frac{1}{k!} (zp)^k \frac{x_{n-k}}{(n-k)!} (zq)^{n-k}. \end{aligned}$$

Thus, using the fact that $x_0 = x_1 = a_0 = a_1 = 0$,

$$\sum_{n=0}^{\infty} \frac{x_n}{n!} z^n = \sum_{n=0}^{\infty} \frac{a_n}{n!} z^n + \sum_{n=0}^{\infty} \left(\sum_{k=0}^{n} \frac{x_k}{k!} (zp)^k \frac{1}{(n-k)!} (zq)^{n-k} + \sum_{k=0}^{n} \frac{1}{k!} (zp)^k \frac{x_{n-k}}{(n-k)!} (zq)^{n-k} \right).$$

Finally, we arrive at

$$x(z) = a(z) + x(zp)e^{zq} + x(zq)e^{zp},$$

where $a(z)$ is the EGF of a_n. The Poisson transform [18, 38] defined as $\tilde{X}(z) = x(z)e^{-z}$ of the above equation is

$$\tilde{X}(z) = \tilde{A}(z) + \tilde{X}(zp) + \tilde{X}(zq), \tag{22}$$

where $\tilde{A}(z) = a(z)e^{-z}$. By analytic depoissonization [18] we expect that $x_n \sim \tilde{X}(n)$ as $n \to \infty$. We refer to Theorem 10.5 of [38] to conclude that this is the case. Thus it remains to find asymptotics of $\tilde{X}(z)$ as $z \to \infty$ along the real axis.

In order to solve asymptotically the functional equation (22), we apply the Mellin transform. The reader is referred to [13, 38] for in-depth discussion of the Mellin transform. In brief, the Mellin transform of a real-valued function $f(x)$ is defined as

$$\mathcal{M}[f(x);s] := f^*(s) = \int_0^\infty f(x)x^{s-1}dx.$$

It is defined in a strip $-\alpha < \Re(s) < -\beta$ when $f(x) = O(x^\alpha)$ for $x \to 0$ and $f(x) = O(x^\beta)$ for $x \to \infty$. Noting that

$$\mathcal{M}[f(ax), s] = a^{-s}f^*(s),$$

we transform the functional equation (22) into the following *algebraic* equation

$$X^*(s) = A^*(s) + p^{-s}X^*(s) + q^{-s}X^*(s),$$

where $X^*(s)$ and $A^*(s)$ are the Mellin transforms of $\tilde{X}(z)$ and $\tilde{A}(z)$, respectively. This leads to

$$X^*(s) = \frac{A^*(s)}{1 - p^{-s} - q^{-s}}$$

for $-2 < \Re(s) < -1$, as easy to see under our assumption on x_0 and x_1. Observe also that

$$A^*(s) = \int_0^\infty \tilde{A}(z)z^{s-1}dz = \sum_{n\geq 2} \frac{a_n}{n!} \int_0^\infty z^n e^{-z} z^{s-1}dz = \sum_{n\geq 2} \frac{a_n}{n!}\Gamma(n+s), \qquad (23)$$

and for $a_n = \lceil \log(n+1) \rceil$ the series converges for $\Re(s) < 0$.

In order to find asymptotics of x_n, we first find the inverse Mellin transform and then depoissonize as in [18, 38]. For this we need to understand zeroes of $1-p^{-s}-q^{-s} = 0$, that is, we study $\mathcal{Z} = \{s \in \mathbb{C} : p^{-s} + q^{-s} = 1\}$. The following lemma is basically due to Schachinger [33] and Jacquet [38] (cf. also [10]).

Lemma 12 *Suppose that* $0 < p < q < 1$ *with* $p + q = 1$, *and let*

$$\mathcal{Z} = \{s \in \mathbb{C} : p^{-s} + q^{-s} = 1\}.$$

Then
(i) *All* $s \in \mathcal{Z}$ *satisfy*

$$-1 \leq \Re(s) \leq \sigma_0,$$

where σ_0 *is a real positive solution of* $1 + q^{-s} = p^{-s}$. *Furthermore, for every integer* k *there uniquely exists* $s_k \in \mathcal{Z}$ *with*

$$(2k-1)\pi/\log(1/p) < \Im(s_k) < (2k+1)\pi/\log(1/p)$$

and consequently $\mathcal{Z} = \{s_k : k \in \mathbb{Z}\}$.

(ii) *If* $\log q/\log p$ *is irrational, then* $s_0 = -1$ *and* $\Re(s_k) > -1$ *for all* $k \neq 0$.

(iii) *If* $\log q/\log p = r/d$ *is rational, where* $\gcd(r,d) = 1$ *for integers* $r, d > 0$, *then* $\Re(s_k) = -1$ *if and only if* $k \equiv 0 \bmod d$. *In particular* $\Re(s_1), \ldots, \Re(s_{d-1}) > -1$ *and*

$$s_k = s_{k \bmod d} + \frac{2(k - k \bmod d)\pi i}{\log p},$$

that is, all $s \in \mathcal{Z}$ *are uniquely determined by* $s_0 = -1$ *and by* $s_1, s_2, \ldots, s_{d-1}$, *and their imaginary parts constitute an arithmetic progression.*

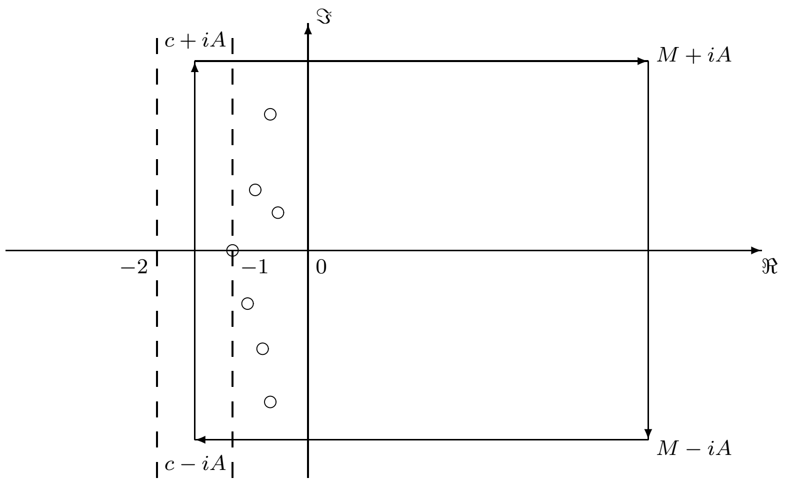


Figure 8: The integration contour (the circles represent zeroes of $p^{-s}+q^{-s}=1$.)

Using this lemma, now we find the asymptotics of $\tilde{X}(z)$ as $z\to\infty$ by the inverse Mellin transform:

$$\tilde{X}(z)=\frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty}X^*(s)z^{-s}ds,$$

where $-2<c<-1$ is a constant. To compute this we apply the standard approach: consider the rectangle $\mathcal{R}$ shown in Figure 8. The integral of $X^*(s)z^{-s}$ along $\mathcal{R}$ is divided into four parts as follows:

$$\lim_{A\to\infty}\int_{\mathcal{R}}=\lim_{A\to\infty}\left(\int_{c-iA}^{c+iA}+\int_{c+iA}^{M+iA}+\int_{M+iA}^{M-iA}+\int_{M-iA}^{c-iA}\right).$$

We are interested in the first integral. We observe that $\tilde{X}(z)$ is infinitely differentiable. Thus the second and the fourth integrals contribute $O(A^{-r})$ for some large r due to the smallness property of Mellin transform (cf. [38]). The contribution of the third integral is computed as follows:

$$\begin{aligned}\left|\int_{M+i\infty}^{M-i\infty}X^*(s)z^{-s}ds\right| &= \left|\int_{+\infty}^{-\infty}X^*(M+it)z^{-M-it}dt\right|\\ &\leq |z^{-M}|\int_{+\infty}^{-\infty}|X^*(M+it)|dt \;=\; O(z^{-M}),\end{aligned}$$

since the integral above exists. Now by the Cauchy residue theorem and Lemma 12, we obtain

$$\frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty}X^*(s)z^{-s}ds+O(z^{-M})=-\sum_{s_k\in\mathcal{Z}}\mathrm{Res}[X^*(s)z^{-s},s=s_k].$$

We observe that the residue at $s_0=-1$ (cf. also [12]). Using these observations and analytic depoissonization [18, 38] we finally conclude the following, proving Lemma 6:
(i) If $\log p/\log q$ is *irrational*, then

$$x_n=\frac{n}{h}A^*(-1)\log e+o(n), \tag{24}$$

where

$$A^*(-1) = \sum_{b\geq 2} \frac{\lceil \log(b+1) \rceil}{b(b-1)}. \tag{25}$$

(ii) If $\log p / \log q = r/d$ (*rational*) with $\gcd(r,d) = 1$, then

$$x_n = \frac{n}{h}\left(A^*(-1) + \Phi(\log_p n)\right)\log e + O\left(n^{1-\eta}\right) \tag{26}$$

for some $\eta > 0$, where

$$\Phi(x) = \sum_{k\neq 0} A^*(-1 + 2k\pi r i/\log p)\exp(2k\pi r x i) \tag{27}$$

is a fluctuating function with a small amplitude. This proves Lemma 6.

Appendix

A Proof of Lemma 2

For completeness, we prove here Lemma 2. It was established in [23] that almost surely one should alter (delete or add) $(2 - o(1))np(1-p)$ edges to obtain a symmetric graph from $G(n,p)$, which is a sufficient condition of our statement. We shall follow the footsteps of [23], except that we derive explicitly the rate of convergence.

First we need some definitions. Let $G = (V,E)$ be a graph and let $\pi : V \to V$ be a permutation of the vertices of G. For a vertex $v \in V$ we define a defect of v with respect to π to be

$$D_\pi(v) = |N(\pi(v)) \ \Delta \ \pi(N(v))|,$$

where $N(v)$ is the set of neighbors of v and Δ denotes the symmetric difference of two sets, that is, $A\Delta B = (A-B)\cup(B-A)$ for two sets A and B. Similarly, we define a defect of G with respect to π to be

$$D_\pi(G) = \max_v D_\pi(v).$$

Finally, we define a defect of a graph G to be

$$D(G) = \min_{\pi\neq identity} D_\pi(G).$$

It is easy to see that a graph G is symmetric if and only if its defect equals zero. Thus we only need to show that $D(G) > 0$ for $G \in \mathcal{G}(n,p)$ with high probability. But we shall next prove that $D(G)$ is at least $(2-o(1))np(1-p)$ with high probability.

Set $\epsilon = \epsilon(n,p)$ such that $\epsilon = o(1)$ and $\epsilon^2 np(1-p) \gg \ln n$. This is possible for all p's satisfying the conditions of the lemma (e.g., $\epsilon = \Theta\left(\sqrt[4]{\frac{\ln n}{np(1-p)}}\right)$.) Fix an arbitrary $2 \leq k \leq n$, and let π be a permutation of vertices of G which fixes all but k vertices. Let U be the set of vertices $\{u | \pi(u) \neq u\}$ and

$$X = \sum_{u\in U} D_\pi(u).$$

By definition, $D_\pi(u)$ is a binomially distributed random variable with expectation either $2(n-2)p(1-p)$ or $2(n-1)p(1-p)$, depending on whether $\pi(\pi(u)) = u$ or not. Therefore,

$$\mathbf{E}[X] = \sum_{u \in U} \mathbf{E}[D_\pi(u)] = (2 - o(1))knp(1-p).$$

We prove next that X is strongly concentrated around its mean, which implies that for some vertex $u \in U$, $D_\pi(u)$ is at least $\mathbf{E}[X]/k$ with high probability. Then we conclude that $D_\pi(G)$ is at least $\mathbf{E}[X]/k$ with high probability by the definition. Finally, we shall prove that, for every possible permutation π, the minimum of $D_\pi(G)$ is still at least $\mathbf{E}[X]/k$ with high probability, which implies that $D(G)$ is at least $\mathbf{E}[X]/k$ with high probability by the definition.

We start with an observation that X depends only on the edges of the graph adjacent to the vertices in U. Moreover, adding or deleting any such edge, say (u, v), can change only the values of at most four terms $D_\pi(u)$, $D_\pi(v)$, $D_\pi(\pi^{-1}(u))$, and $D_\pi(\pi^{-1}(v))$ in the sum, each by at most 1. Here X is a random variable on a probability space generated by a finite set of mutually independent 0/1 choices, indexed by i. Let p_i be the probability that choice i is 1, and let c be a constant such that changing any choice i (keeping all other choices the same) can change X by at most c. Set $\sigma^2 = c^2 \sum_i p_i(1-p_i)$. In [3] it is shown that for all positive $t < 2\sigma/c$,

$$P(|X - \mathbf{E}[X]| > t\sigma) \le 2e^{-t^2/4}.$$

In this case, $c = 4$ and $\sigma^2 = 16(\binom{n}{2} - \binom{n-k}{2})p(1-p) = \Theta(knp(1-p))$. Therefore, for some positive constant α,

$$P(|X - \mathbf{E}[X]| > \epsilon knp(1-p)) \le e^{-\alpha\epsilon^2 knp(1-p)}.$$

Thus, with probability at least $1 - e^{-\alpha\epsilon^2 knp(1-p)}$, there is a vertex in U with defect at least

$$\frac{1}{k}(\mathbf{E}[X] - \epsilon knp(1-p)) = (2 - o(1))np(1-p).$$

Therefore,

$$P(D_\pi(G) \le (2-\epsilon)np(1-p)) \le e^{-\alpha\epsilon^2 knp(1-p)} = P_k.$$

Now we see that the number of permutations which fixes $n-k$ vertices is at most $\binom{n}{k}k!$. Therefore, the probability that there exists a permutation such that the defect of G with respect to it is less than $(2-\epsilon)np(1-p)$ is at most

$$\begin{aligned}
\sum_{k=2}^{n} \binom{n}{k} k! P_k &\le \sum_{k=2}^{n} n^k e^{-\alpha\epsilon^2 knp(1-p)} \\
&= \sum_{k=2}^{n} \left(e^{-\alpha\epsilon^2 np(1-p) + \ln n}\right)^k \\
&\le \beta\left(e^{-\alpha\epsilon^2 np(1-p) + \ln n}\right)^2 \quad \text{for some constant } \beta \ (\because \epsilon^2 np(1-p) \gg \ln n) \\
&< \beta\left(e^{-\gamma \ln n + \ln n}\right)^2 \quad \text{for any positive constant } \gamma \ (\because \epsilon^2 np(1-p) \gg \ln n) \\
&= \beta\left(n^{1-\gamma}\right)^2 = O\left(n^{-w}\right) \quad \text{for any positive constant } w.
\end{aligned}$$

The last equality is obtained by setting $w = 2\gamma - 2$ and choosing $\gamma > 1$. Therefore, the probability that G is symmetric is at most $O\left(n^{-w}\right)$ for any positive constant w.

References

[1] M. Adler and M. Mitzenmacher, Towards compressing web graphs, *In Proc. of the IEEE Data Compression Conference*, 203–212, 2001.

[2] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Dover, New York, 1964.

[3] N. Alon, J.H. Kim, and J.H. Spencer, Nearly perfect matchings in regular simple hypergraphs, *Israel J. Math*, 100, 171–187, 1997.

[4] B. Bollobas, *Random Graphs*, Cambridge University Press, Cambridge, 2001.

[5] F.P. Brooks Jr, Three great challenges for half-century-old computer science, *Journal of the ACM*, 50(1), 25–26, 2003.

[6] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On Compressing Social Networks, *Proc. ACM KDD*, 2009.

[7] Y. Choi and W. Szpankowski, Compression of graphical structures, *IEEE International Symposium on Information Theory*, Seoul, 364–368, 2009.

[8] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 2006.

[9] M. Drmota, H.-K. Hwang, and W. Szpankowski, Precise average redundancy of an idealized arithmetic coding, *Proc. Data Compression Conference*, 222-231, 2002.

[10] M. Drmota, Y. Reznik, and W. Szpankowski, Tunstall Code, Khodak Variations, and Random Walks, preprint 2009.

[11] P. Erdős and A. Rényi, Asymmetric graphs, *Acta Math. Acad. Sci. Hungar.* 14, 295–315, 1963.

[12] G. Fayolle, P. Flajolet, and M. Hofri, On a Functional Equation Arising in the Analysis of a Protocol for a Multi-Access Broadcast Channel, *Advances in Applied Probability*, 18(2), 441–472, 1986.

[13] P. Flajolet and R. Sedgewick, *Analytic Combinatorics*, Cambridge University Press, Cambridge, 2008.

[14] F. Harary and E.M. Palmer, *Graphical Enumeration*, Academic Press, 1973.

[15] F. Harary, E.M. Palmer, and R.C. Read, The number of ways to label a structure, *Psychometrika*, 32(2), 155–156, 1967.

[16] M. Hassani, Approximation of the dilogarithm function, *J. Inequalities in Pure and Applied Mathematics*, 8, 1–7, 2007.

[17] P. Jacquet and W. Szpankowski, Asymptotic behavior of the Lempel-Ziv parsing scheme and digital search trees, *Theoretical Computer Science*, 144(1&2), 161–197, 1995.

[18] P. Jacquet, and W. Szpankowski, Analytical depoissonization and its applications, *Theoretical Computer Science*, 201, 1–62, 1998.

[19] P. Jacquet, and W. Szpankowski, Entropy computations via analytic depoissonization, *IEEE Trans. on Information Theory*, 45, 1072–1081, 1999.

[20] Y. Jia, E.-H. Yang, D.-K. He, and S. Chan, A greedy renormalization method for arithmetic coding, *IEEE Transactions on Communications*, 55(8):1494–1503, 2007.

[21] J.C. Kieffer, A survey of advances in hierarchical data compression, Technical Report, Dept. of Electrical & Computer Engineering, University of Minnesota, 2000.

[22] J. Kieffer, E-H. Yang, and W. Szpankowski, Structural Complexity of Random Binary Trees *2009 International Symposium on Information Theory*, 635-639, Seoul, 2009.

[23] J.H. Kim, B. Sudakov, and V.H. Vu, On the asymmetry of random regular graphs and random graphs, *Random Structures and Algorithms*, 21(3-4), 216–224, 2002.

[24] C. Knessl, Integral representations and asymptotic expansions for Shannon and Renyi entropies, *Appl. Math. Lett.*, 11, 69–74, 1998.

[25] B. MacArthur, Sanchez-Garcia R, and J. Anderson. Symmetry in complex networks. *Discrete Applied Mathematics*, 156, 18, 3525-3531, 2008.

[26] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat. Systematic topology analysis and generation using degree correlation. In *SIGCOMM*, Pisa, 2006.

[27] M. Naor, Succinct representation of general unlabeled graphs, *Discrete Applied Mathematics*, 28(3), 303–307, 1990.

[28] L. Peshkin, Structure induction by lossless graph compression, *In Proc. of the IEEE Data Compression Conference*, 53–62, 2007.

[29] N. Rashevsky. Life, information theory, and topology. *Bull. Math. Biophysics*, 17:229–235, 1955.

[30] J. Rissanen, Complexity and Information Data, in *Entropy*, (eds. A. Graven, G. Keller and G. Warnecke), Princeton University Press, 2003.

[31] S. Ross, *Stochastic Processes*, John Wiley & Sons, New York, 1983.

[32] S. A. Savari, Compression of words over a partially commutative alphabet, *IEEE Transactions on Information Theory*, 50, 1425-1441, 2004.

[33] W. Schachinger, Limiting distributions for the costs of partial match retrievals in multidimensional tries. *Random Structures and Algorithms*, 17(3-4), 428–459, 2000.

[34] C.E. Shannon, A mathematical theory of communication, *Bell System Technical Journal*, 27, 379–423 and 623–656, 1948.

[35] C. Shannon. The lattice theory of information. *IEEE Transaction on Information Theory*, 1:105–107, 1953.

[36] G. Simonyi, Graph Entropy: A Survey, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 2000.

[37] J. Sun, E.M. Bollt, and D. Ben-Avraham, Graph compression–save information by exploiting redundancy, *Journal of Statistical Mechanics: Theory and Experiment*, P06001, 2008.

[38] W. Szpankowski, *Average Case Analysis of Algorithms on Sequences*, John Wiley & Sons, New York, 2001.

[39] E. Trucco. A note on the information content of graphs. *Bull. Math. Biophysics*, 18:129–135, 1956.

[40] Gy. Turan, On the succinct representation of graphs, *Discrete Applied Mathematics*, 8(3), 289–294, 1984.

[41] F.M.J. Willems, Y.M. Shtarkov, and T.J. Tjalkens, The Context Tree Weighting Method: Basic Properties, *IEEE Transactions on Information Theory*, 41, 653–664, 1995.