Practical Random Tree Generation using Spanning Trees: Entropy and Compression

Anonymous Authors[1]

Abstract

Tree structures make an appearance in many learning-related problems, most importantly in Graph Neural Networks. Modeling and simulating the appearance of these data structures can be done using random tree generators. However, there has been very little study on random models that are able to capture the dynamics of networks. We introduce the random spanning tree model, which is a random tree generator that is based on generating a tree from an already existing network topology. The Shannon entropy of this model is then analysed, and upper bounds to it are found. As compression can be beneficial because of the complexity of large trees, we then introduce a universal approach to compressing trees generated using the spanning tree model. It will be shown that the proposed method of compression introduces a redundancy that tends to zero for larger trees.

1. Introduction

Despite their vast applications, there are very few models for the random generation of trees. A good random tree generation model can be used to model and simulate many real-world phenomena, especially in learning applications. The existing models for trees are very limited, and most of them rely solely on a uniform distribution among the possible trees. For example, random generation of a Prüfer sequence (Prüfer, 1918) can result in a random tree. Other models focus only on specific type of tree, such as binary trees (Mäkinen, 1999). One of the most detailed studies on random trees can be found in (Drmota, 2009), where several random tree models are introduced and analysed. The models that are analysed in (Drmota, 2009) include Polya trees (Mauldin et al., 1992), Galton-Watson trees (Watson & Galton, 1875), and the simply generated tree model (Drmota, 2009). However, these models come with their own drawbacks. For instance, the number of nodes in the generated trees can grow indefinitely, and the size of the trees that we need can not be set using a parameter in the model.

Because of these reasons, we introduce a novel random tree generator in this paper. The introduced model, which we call the spanning tree model, is built based on what usually happens in practice. Random trees are often generated from an underlying network topology, as one of its spanning trees. Spanning trees of an underlying network topology have huge applications in areas such as decision trees in machine learning and random forest classifiers (Ho, 1995). The introduced model is not only based on practical scenarios, but it is also very flexible for simulating different situations as its different parameters can be adjusted to fit many real-world scenarios.

After having introduced the random tree source, we move on to study its information-theoretic parameters. This analysis is performed because of the fact that trees, and graphical data structures in general, become overly complex as the number of nodes grows. Therefore, we aim to quantify the complexity of the introduced source. Additionally, these trees need to be stored and/or communicated through a communication channel in practice. For this reason, we will also study methods for optimal compression of said trees. The compression will be studied for single trees rather than a sequence of trees, specifically as the number of nodes grows large. This is because in practical scenarios we face single trees more often than a sequence of trees, and compression is mainly needed when complexity grows with the number of nodes.

The rest of the paper is organised as follows. Section 2 introduces the spanning tree model. In section 3, we study the bounds on the entropy of the spanning tree model. Section 4 proposes a compression method for optimal compression of trees generated from a specific class of the spanning tree model. Ultimately, the paper is concluded and future directions of research are proposed.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the Neural Compression Workshop @ ICML 2023. Do not distribute.

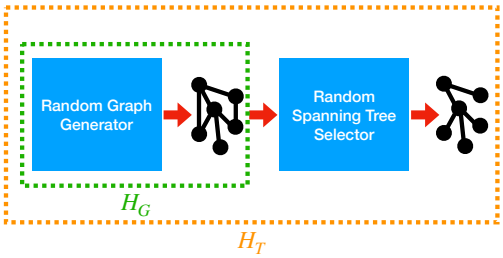

Figure 1. Steps of the spanning tree model

2. The Spanning Tree Model

In this section, we will introduce a novel random tree generation model called the spanning tree model. In practical applications, the trees we work with are often a spanning tree of a network. An example of this can be seen in network routing. Routing tables are usually used to store the shortest paths from any node in a network to any other one. It can be shown that the routing table for a node in a network is essentially representing a rooted tree, with the root being the origin node. It can also be seen that if the network forms a connected graph, this rooted tree is a spanning tree of the underlying network. Therefore, selecting a random spanning tree of the underlying network can be a practical model for generating random trees. In this section, we will introduce our proposed random tree generator, called the spanning tree model. We will use the following definitions for this purpose.

Definition 2.1 (Random Graph Source)**.** A random graph source consists of a set G, which includes all the possible graphs that can be generated by the source, alongside a probability distribution $p_G(g)$ defined on the set G, which shows the probability of individual graphs being generated by the source.

Definition 2.2 (Random Tree Source)**.** A random tree source consists of a set T, which includes all the possible trees that can be generated by the source, alongside a probability distribution $p_T(t)$ defined on the set T, which shows the probability of individual graphs being generated by the source.

Assume that we have a random graph source G. The first step is to create a graph g, according to the distribution of G. g will then have a number of spanning trees (which can also be zero). We then randomly choose one of the spanning trees of g as the generated tree. This random choice can be done according to any arbitrary distribution, which can also be dependent on g. This is the basis of a general spanning tree model. Fig. 1 shows the steps of the spanning tree model.

It is clear that the spanning tree model is a very general model, as the random graph generator and the way a spanning tree is selected are both distributions that can be chosen according to the network that we want to simulate. To provide a good example of how these parameters can be chosen, we introduce ER random spanning trees. For the random graph generator of this model, we use The ER model (Gilbert, 1959). The ER model is known as one of the most simple, yet effective, random graph generators. It has been shown to be effective in simulating real-world phenomena such as epidemics and evolutionary conflicts (Cannings & Penman, 2003). Additionally, for choosing a random spanning tree from the created graph, we simply choose one of its spanning trees in a uniform manner. We will study this model more in the following sections.

3. Entropy of the spanning tree model

In this section, we will attempt to quantify the entropy of the spanning tree model. We use Shannon's entropy (Shannon, 1948) for this purpose. All the logarithms are calculated in base two, and therefore the resulting entropy is in terms of bits. We will use the following notations in our calculations.

- H_G: Entropy of the random graph source
- H_T: Entropy of the spanning tree model source

The goal in this section is to find H_T, assuming that H_G is known or can be easily calculated. We can write the following equation to find H_T.

$$H_T = H_G + H(T|G) - H(G|T) \tag{1}$$

If we assume that the output trees are chosen uniformly from the spanning trees of the graph, then we can write the following equation for calculating $H(T|G)$.

$$H(T|G) = \sum_{g \in G} p_G(g) H(T|G = g) \tag{2a}$$

$$= \sum_{g \in G} p_G(g) \log_2 s(g), \tag{2b}$$

where the sum is taken among all the connected graphs that can be generated using the random graph generator, p_G shows the probability distribution of the graph source, and $s(g)$ shows the number of spanning trees of graph g. The need for connectivity of the graphs arises from the fact that they need to have at least one spanning tree for $\log_2 s(g)$ to be defined.

Eq. (1) and (2b) show that to calculate the entropy of the spanning tree model, we need knowledge about the underlying distribution for the network topology, as well as the

number of spanning trees that exists for each graph. Unfortunately, the number of spanning trees of a graph, often called its tree-number, is not easy to find. The entropy also depends on the model that is used to generate the underlying graph topology, which is not always known. Because of these limitations, we will focus on ER Spanning Trees, which were introduced in the previous section.

In ER Spanning Trees, the underlying network topology is created using the ER model. In an ER graph, each edge will be present with a probability of p, independent of other edges. As the ER model does not guarantee connectivity, there will be some graphs that do not even have a spanning tree. Additionally, it is known that there is no closed-form formula to calculate the number of spanning trees of a graph given its number of nodes and edges. Because of these reasons, we will find an upper bound to the entropy of the spanning trees of ER graphs rather than its actual value.

We consider ER graphs with n nodes, with parameter p. It can easily be shown that the entropy of the graphs created using this model can be calculated using the following equation.

$$H_G = \binom{n}{2} H(p), \tag{3}$$

where

$$H(p) = -p \log_2 p - (1-p) \log_2 (1-p).$$

Additionally, we assume that once an ER graph is created, one of its spanning trees is chosen uniformly. If the graph does not have any spanning trees, then simply no tree is chosen. To find an upper bound to the entropy of the spanning trees created this way, we use the Grimmett upper bound formula (Bozkurt, 2012). Grimmett's formula gives us the following upper bound to the number of the spanning trees of graph g.

$$s(g) \leq \frac{1}{n} \left(\frac{2e(g)}{n-1} \right)^{n-1}, \tag{4}$$

Using this upper bound, we can prove the following proposition. The proof can be found in appendix A.

Proposition 3.1. *The entropy of ER spanning trees is upper bounded based on the following inequality.*

$$H_T \leq (n-1) \left(H(p) + \log_2 (np) \right) \tag{5}$$

Proposition 3.1 gives us an upper bound on the entropy of trees created using the ER Spanning Tree model. Fig. 2 illustrates this entropy, and compares it with the entropy of the graph, and the maximum entropy for trees. The maximum entropy is calculated using the fact that a uniform distribution maximises the entropy, and there exist n^{n-2} possible labelled trees on n nodes. The simulation is run for

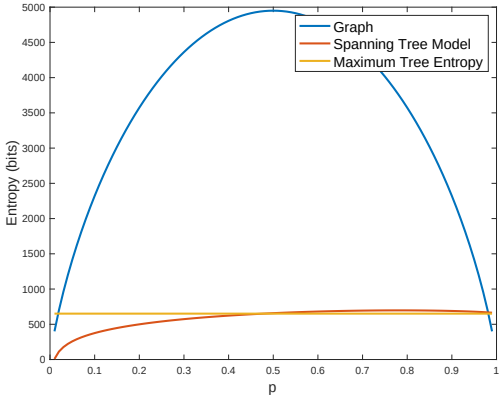

Figure 2. Entropy upper bound for ER spanning trees for graphs with 100 nodes as a function of the ER parameter p

ER graphs with 100 nodes, and the entropy is plotted as a function of the ER parameter p. It can be seen that for larger values of p, the estimated upper bound for the entropy is larger than the maximum entropy. However, (5) is providing us with a tighter upper bound when used for lower values of p.

4. Compression of ER Spanning Trees

In this section, we will introduce a universal and optimal compression algorithm for the ER spanning tree model. Before doing so, we need to define what we mean by universality and optimality in the context of these methods.

Optimality: It was shown by Shannon (Shannon, 1948) that the entropy of a random variable provides a lower bound on the average code length that we can use to compress that random variable. Therefore, we are looking for a compression algorithm whose average codeword length is close enough to the entropy of the random tree source at hand.

Universality: Models for generating random trees have specific parameters that need to be set. For instance, the random graph generator and its parameters need to be set when using the introduced spanning tree model. By looking for universal compression algorithms for specific families of random trees, we are essentially looking for compression algorithms that perform optimally regardless of the model parameters. For example, if we develop a universal compression algorithm for ER random spanning trees, we want it to perform optimally regardless of the ER parameter p.

The idea of all existing universal compression algorithms is that as the parameter of the distribution is unknown, the distribution needs to be somehow learned from the data. For instance, the renowned family of Lempel-Ziv (Ziv & Lempel, 1977) compression algorithms, uses dictionaries to store and learn the most common patterns that can happen

for the random variable at hand. However, this demands having a sequence of the random variable, so that the most common patterns can be learned. Whereas, in this paper, we are interested in compressing single large trees, rather than a sequence of them. In this case, optimality translated into the average codeword length for all possible trees to be close enough to the entropy of the source. If we use $E_{L,n}$ to show the expected codeword length for trees with n nodes and H_n to show the entropy of the tree source for trees with n nodes, our goal is for the compressor to satisfy

$$\lim_{n\to\infty} E_{L,n} = H_n, \tag{6}$$

to ensure that our compression algorithm is asymptotically optimal on single trees when n and consequently the need for compression grow. In order to achieve the condition in (6) and still have a universal compression algorithm, our main approach will be to decompose each single tree into a sequence of other random variables, and then apply existing universal compression algorithms to those sequences.

We propose the following approach for coding ER spanning trees. We divide our coding process into two steps: extracting certain bits from the adjacency matrix of the tree, and then compressing the extracted sequence of bits. We first start by describing the bit extraction process.

Bit extraction: We start by looking at the adjacency matrix of the tree, starting with the row corresponding to the connections of node 1. This row consists of $n-1$ bits $(a_{1,2}, a_{1,3} \ldots, a_{1,n})$, where each bit represents the existence of a connection between node 1 and the other nodes in the tree. We take all these bits during the extraction process. After this, we know all the connections of node 1 in the tree. Let us show the number of these connections with random variables C_1. Each pair among these C_1 edges removes the possibility of having one other edge in the tree. For instance, if node 1 is connected to both nodes i and j, there can not be a connection between i and j in the tree. Therefore, having the connections of node 1 removes the need for including $\binom{C_1}{2}$ bits in the adjacency matrix of the tree, and we will know exactly which ones. After having described the connections of node 1, we go to the nodes to which node 1 is connected in the order of their labels. We continue by writing down the rows of the adjacency matrix corresponding to these rows, without the connections that have been covered before or whose state is known due to the described edge elimination process. This way, the second node requires less than impossibility of this node being connected to other neighbours of node 1. We can carry on this way and explain the remaining connections of each node, until all the nodes in the tree are covered. Note that bit extraction can be applied to any simple graph, and not just ER graphs. We show the process of bit extraction from a tree t with $f(t)$.

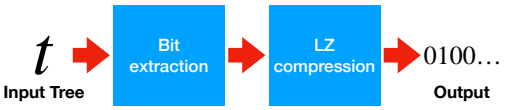

Figure 3. Proposed algorithm for compressing ER Spanning Trees

After having extracted certain bits of the adjacency matrix using f, we simply feed them into a universal compression algorithm, such as the Lempel-Ziv-Welch algorithm (Welch, 1984). Fig. 3 summarizes our proposed compression technique.

Proposition 4.1. *The redundancy of the proposed compression algorithm tends to zero as n goes to infinity.*

Proposition 4.1 is proven in appendix B. It is also shown that it tends towards zero regardless of the value of p and the way that the random spanning trees were chosen. Therefore, it can be said that the proposed method is universal in the sense that it does not depend on the ER parameter or the random tree selection process. Note that the proposed compression algorithm can be further generalized by generalizing the bit extraction process. For instance, the process does not necessarily need to start from node 1, and a DFS traversal would have worked too. We aim to work on a generalization of the traversal methods that can be used for this purpose in the future.

5. Conclusion

In this paper, we discussed the need for having novel random tree generators to be used in practical scenarios that involve tree data structures. The random spanning tree model was introduced as a simple, yet practical, method for generating random trees. It was shown how this model can be adapted to different scenarios by choosing the appropriate random graph generation model, and the probability distribution for selecting a random spanning tree. We then introduced ER spanning trees, as it is known that the ER model has many practical applications in network science. We continued by analysing the entropy of the proposed models, and measuring their complexity. Having calculated the entropy of the ER spanning tree model as a lower bound for its compression, we moved on to introduce a universal compression algorithm for this family of trees. It was shown that our proposed compression algorithm will achieve a redundancy of zero for large trees, which are the reason we need compression for these structures in the first place. To continue this work, we firstly aim to consider other tree traversal algorithms for the bit extraction process, such that a query-preserving compression algorithm can be designed. Additionally, we aim to study the application of the proposed compression method on network routing protocols.

References

Bozkurt, Ş. B. Upper bounds for the number of spanning trees of graphs. *Journal of Inequalities and Applications*, 2012(1):1–7, 2012.

Cannings, C. and Penman, D. Ch. 2. models of random graphs and their applications. In *Stochastic Processes: Modelling and Simulation*, volume 21 of *Handbook of Statistics*, pp. 51–91. Elsevier, 2003. doi: https://doi.org/10.1016/S0169-7161(03)21004-X. URL https://www.sciencedirect.com/science/article/pii/S016971610321004X.

Drmota, M. *Random trees: an interplay between combinatorics and probability*. Springer Science & Business Media, 2009.

Gilbert, E. N. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

Ho, T. K. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pp. 278–282. IEEE, 1995.

Jensen, J. L. W. V. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta mathematica*, 30(1):175–193, 1906.

Mäkinen, E. Generating random binary trees—a survey. *Information Sciences*, 115(1-4):123–136, 1999.

Mauldin, R. D., Sudderth, W. D., and Williams, S. C. Polya trees and random distributions. *The Annals of Statistics*, pp. 1203–1221, 1992.

Plotnik, E., Weinberger, M. J., and Ziv, J. Upper bounds on the probability of sequences emitted by finite-state sources and on the redundancy of the lempel-ziv algorithm. *IEEE transactions on information theory*, 38(1): 66–72, 1992.

Prüfer, H. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys*, 27(1918):742–744, 1918.

Shannon, C. E. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Watson, H. W. and Galton, F. On the probability of the extinction of families. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 4:138–144, 1875.

Welch, T. A. A technique for high-performance data compression. *Computer*, 17(06):8–19, 1984.

Ziv, J. and Lempel, A. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

A. Proof of Proposition 3.1

We first start by finding an upper bound to $H(T|G)$ for the ER model. Notice that in the calculations below, we might be faced with graphs that do not even have a spanning tree to choose from. Note that Grimett's formula has already accounted for those instances by providing us with an upper bound. For the exceptional case that $e(g) = 0$, we will be faced with $\log_2 0$ in our calculations. To avoid this, as we are interested in an upper bound, we assume that these graphs also have at least one spanning tree. Therefore, $\log_2 0$ is automatically replaced with $\log_2 1 = 0$ whenever it is observed. We will get the following inequality by plugging (4) into (2b).

$$\begin{aligned} H(T|G) &\leq \sum_g p_G(g) \log_2 \left(\frac{1}{n} \left(\frac{2e(g)}{n-1} \right)^{n-1} \right) \\ &= -\log_2 n \sum_g p_G(g) \\ &\quad - (n-1) \sum_g p_G(g) \log_2 (n-1) \\ &\quad + (n-1) \sum_g p_G(g) (\log_2 e(g) + 1) \\ &= -\log_2 n - (n-1) \log_2 (n-1) \\ &\quad + (n-1) \sum_g p_G(g) \log_2 e(g) + (n-1). \end{aligned} \tag{7}$$

To calculate the term $\sum_{g \in G} p(g) \log_2 e(g)$ in (7), we can take the sum over the number of possible edges in the graph and write

$$\sum_{g \in G} p(g) \log_2 e(g) = \sum_{i=0}^{\binom{n}{2}} \binom{\binom{n}{2}}{i} p^i (1-p)^{\binom{n}{2}-i} \log_2 i. \tag{8}$$

Using Jensen's inequality (Jensen, 1906), we can find an upper bound to (8).

$$\begin{aligned} &\sum_{i=0}^{\binom{n}{2}} \binom{\binom{n}{2}}{i} p^i (1-p)^{\binom{n}{2}-i} \log_2 i \\ &\leq \log_2 \sum_{i=0}^{\binom{n}{2}} \binom{\binom{n}{2}}{i} p^i (1-p)^{\binom{n}{2}-i} i \\ &= \log_2 \left(\binom{n}{2} p \right) \end{aligned} \tag{9}$$

By plugging in the results of (8) and (9) into (7), we will get the following upper bound for $H(T|G)$.

$$\begin{aligned} H(T|G) &\leq (n-1) \log_2 \left(\binom{n}{2} p \right) \\ &\quad - \log_2 n - (n-1) \log_2 (n-1) + n - 1 \\ &\leq (n-1) \left(\log_2 \left(\binom{n}{2} p \right) - \log_2 \frac{(n-1)}{2} \right) \\ &= (n-1) \log_2 (np) \end{aligned} \tag{10}$$

We now move on to calculate the term $H(G|T)$ in (1). Notice that given a spanning tree of an ER graph with n nodes, we will know the status of $n-1$ edges out of the possible $\binom{n}{2}$ edges of the graph. Therefore, given a spanning tree of the graph,

the remaining entropy is simply the entropy of the remaining $\binom{n}{2} - (n-1)$ edges of an ER graph. Consequently, we can write the following equation.

$$\begin{aligned} H(G|T) &= \sum_t p_T(t) H(G|T=t) \\ &= \sum_t p_T(t) \left(\binom{n}{2} - (n-1) \right) H(p) \\ &= \left(\binom{n}{2} - (n-1) \right) H(p) \sum_t p_T(t) \\ &= \left(\binom{n}{2} - (n-1) \right) H(p), \end{aligned} \tag{11}$$

where the sum is taken over all possible spanning trees on the n nodes of the graph, and p_T shows the probability distribution of the trees.

Ultimately, we insert the results from (3), (10), and (11) into (1) to get the total upper bound on the entropy of the spanning trees of the ER model. This will provide us with the following inequality after simplification.

$$H_T \leq (n-1) \left(H(p) + \log_2(np) \right) \tag{12}$$

B. Proof of Proposition 4.1

In this section, we quantify the redundancy of the proposed compression algorithm. We will perform the calculations for the case where LZ78 is used as the universal compression. However, the result is similar for other compression algorithms in the LZ family. It is shown in (Plotnik et al., 1992) that for a binary sequence of length l, the redundancy of LZ78, which we show with $\mathscr{R}$, satisfies the following inequality.

$$\mathscr{R} \leq \frac{\ln \ln l}{\ln l} + \mathcal{O}\left(\frac{1}{\ln l} \right) \tag{13}$$

Therefore, if we use $L(t)$ to show the length of $f(t)$, we can use the following inequality to find the average redundancy of the proposed compression algorithm.

$$\mathbb{E}\left[\mathscr{R}\right] \leq \mathbb{E}\left[\frac{\ln \ln L(t)}{\ln L(t)} \right] + \mathbb{E}\left[\mathcal{O}\left(\frac{1}{\ln L(t)} \right) \right] \tag{14}$$

As it can be easily shown that $\ln \ln x / \ln x$ is a concave function, we can use Jensen's inequality (Jensen, 1906) for the first term in (14) and write

$$\mathbb{E}\left[\frac{\ln \ln L(t)}{\ln L(t)} \right] \leq \frac{\ln \ln \mathbb{E}[L(t)]}{\ln \mathbb{E}[L(t)]}. \tag{15}$$

Based on (15), we need to calculate $\mathbb{E}[L(t)]$. Note that the bit extraction process induces an order on the nodes of tree based on the traversal order. Looking closely, this is simply a Breadth-First Search traversal of the tree, by choosing node 1 as the root. Let us show the number of connections left to be described for the ith node in this sequence using A_i. Firstly, we know that $A_1 = n - 1$. Going to each new node, the need for describing the connections to all the nodes that came before it and all their neighbours is removed. As in each step, we do not have any prior information about the connections to the remaining nodes, each bit can be considered an independent Bernoulli process just like in the original graph. Therefore, for $i > 1$ the expected value of unknown connections is $n - 1$, minus the $i - 1$ node that have come before and their expected number of edges, which is simply p times their number of expected connections. However, we must take into account that this way we are double counting all the prior nodes except for node 1, and this needs a correction term. Based on these reasons, we can write the following recursive equations for the expected values of A_is.

$$\begin{cases} \mathbb{E}[A_1] = n-1 \\ \mathbb{E}[A_i] = n - i - p\sum_{j=1}^{i-1}\mathbb{E}[A_j] + (i-2)p, \quad i > 1 \end{cases} \tag{16}$$

Eq. (16) will result in the following recursive equation for $\mathbb{E}[A_i]$ for $i > 1$.

$$\begin{cases} \mathbb{E}[A_1] = n-1 \\ \mathbb{E}[A_i] = (1-p)\mathbb{E}[A_{i-1}] - 1 + p, \quad i > 1 \end{cases} \tag{17}$$

Solving (17) gives us the following solution.

$$\mathbb{E}\left[A_i\right] = \frac{(p-1)^2 - ((n-2)p+1)\,(1-p)^i}{p(p-1)} \tag{18}$$

Eq. (18) gives us the following result for the expected number of bits to code.

$$\sum_{i=1}^{n}\mathbb{E}[A_i] = \frac{\left(np^2 - (1-p)^n - p\left(n(1-p)^n - 2(1-p)^n + 2\right) + 1\right)}{p^2} \tag{19}$$

If we use $h(n)$ to show the term calculated in (19), we will only need to code a maximum of $h(n)$ bits from the adjacency matrix of the tree on average. We can write the following equation by inserting (19) into (15).

$$\mathbb{E}[\frac{\ln\ln L(t)}{\ln L(t)}] \leq \frac{\ln\ln h(n)}{\ln h(n)} \tag{20}$$

For the $\mathbb{E}\left[\mathcal{O}\left(\frac{1}{\ln L(t)}\right)\right]$ term in (14), we can simply replace it with a coefficient of $\mathbb{E}[\frac{\ln\ln L(t)}{\ln L(t)}]$ and the inequality will still hold. Therefore, we will have the following upper bound on the average redundancy of the compression algorithm.

$$\mathbb{E}[\mathscr{R}] \leq K\frac{\ln\ln h(n)}{\ln h(n)}, \tag{21}$$

where K is a constant. It can easily be seen that

$$\lim_{n\to\infty} h(n) = n. \tag{22}$$

Therefore, we can write

$$\lim_{n\to\infty}\mathbb{E}[\mathscr{R}] = \lim_{n\to\infty} K\frac{\ln\ln h(n)}{\ln h(n)} = 0. \tag{23}$$