Automata and Graph Compression

Mehryar Mohri
Courant Institute and Google Research
mohri@cims.nyu.edu

Michael Riley
Google Research
riley@google.com

Ananda Theertha Suresh
UCSD
asuresh@ucsd.edu

***Abstract*—We present a theoretical framework for the compression of automata, which are widely used representations in speech processing, natural language processing and many other tasks. As a corollary, our framework further covers graph compression. We introduce a probabilistic process of graph and automata generation that is similar to stationary ergodic processes and that covers real-world phenomena. We also introduce a universal compression scheme LZA for this probabilistic model and show that LZA significantly outperforms other compression techniques such as *gzip* and the UNIX *compress* command for several synthetic and real data sets.**

***Index Terms*—Lempel-Ziv, universal compression, graphs**

I. Introduction

Massive amounts of data are generated and processed by modern search engines and popular online sites, which have been reported to be in the order of hundreds of petabytes. At a different level, sophisticated models are commonly stored on mobile devices for tasks such as speech-to-text conversion. Thus, efficient storage and compression algorithms are needed both at the data warehouse level (petabytes of data) and at a device level (megabytes of data). Memory and storage constraints for the latter, as well as the communication costs of downloading complex models, further motivate the need for effective compression algorithms.

Most existing compression techniques were designed for sequential data. They include Huffman coding and arithmetic coding, which are optimal compression schemes for sequences of symbols distributed i.i.d. according to some unknown distribution [10], or the *Lempel-Ziv* schemes, which are asymptotically optimal for stationary ergodic processes [15], [20], or various combinations of these schemes, such as the UNIX command *compress*, an efficient implementation of Lempel-Ziv-Walsh (LZW) and *gzip*, a combination of Lempel-Ziv-77 (LZ77) and Huffman coding.

However, much of the modern data processed at the data warehouse or mobile device levels admits further structure: it may represent segments of a very large graph such as the web graph or a social network, or include very large finite automata or finite-state transducers such as those widely used in speech recognition or in a variety of other language processing tasks such as machine translation, information extraction, and tagging [17]. These data sets are typically very large. Web graphs contain tens of billions of nodes (web pages). In speech processing, a large-alphabet language model may have billions of word edges. These examples and applications strongly motivate the need for structured data compression in practice.

But, how can we exploit this structure to devise better compression algorithms? Can we improve upon the straightforward serialization of that data followed by the subsequent application of an existing sequence compression algorithm? Surprisingly, the problem of compressing such structured data has received little attention. Here, we precisely study the problem of structured data compression. Motivated by the examples just mentioned, we focus on the problem of automata compression and, as a corollary, graph compression.

An empirical study of web graph compression was given by [5], [6], [14]. A theoretical study of the problem was first presented by [1] who proposed a scheme using a *minimum spanning tree* to find similar nodes to compress. However, the authors showed that many generalizations of their problem are NP-hard. Motivated by probabilistic models, [8], [9] later showed that arithmetic coding can be used to nearly optimally compress (the *structure* of) graphs generated by the Erdős-Rényi model.

An empirical study of automata compression has been presented by several authors in the past, including [11], [12]. However, we are not aware of any theoretical work focused on automata compression. Our objective is three-fold: (i) propose a probabilistic model for automata that captures real-world phenomena; (ii) provide a provably universal compression algorithm; and (iii), show experimentally that the algorithm fares well compared to techniques such as gzip and compress. Note that the probabilistic model we introduce can be viewed as a generalization of Erdős-Rényi graphs [4].

The rest of the paper is organized as follows: in Section II, we briefly describe finite automata and some of their key properties. In Section III, we describe our probabilistic model and show that it helps cover many real-world applications. In Section IV, we describe our proposed algorithm, LZA, and prove its optimality. In Section V, we further demonstrate our algorithm's effectiveness in terms of its degree of compression. Due to space constraints, all the proofs are deferred to an extended version of the paper.

II. Directed Graphs and Finite Automata

A finite automaton A is a 5-tuple $(Q, \Sigma, \delta, q_I, F)$ where $Q = \{1, 2, \ldots, n\}$ is a finite set of states, $\Sigma = \{1, 2, \ldots, m\}$ a finite alphabet, $\delta\colon Q \times \Sigma \to Q^*$ the transition function, $q_I \in Q$ an initial state, and $F \subseteq Q$ the set of final states. Thus, $\delta(q, a)$ denotes the set of states reachable from state q via transitions labeled with $a \in \Sigma$. If there is no transition by label a, then $\delta(q, a) = \emptyset$. We denote by $E \subseteq Q \times \Sigma \times Q$ the set of all transitions (q, a, q') and by $E[q]$ the set of all transitions from state q. With the automata notation just introduced, a directed graph can be defined as a pair (Q, δ) where $Q = \{1, 2, 3, \ldots, n\}$ is a finite set of nodes and $\delta\colon Q \to Q^*$ a finite set of edges where, for any node q, $\delta(q)$ denotes the set of nodes connected to q.

An example of an automaton is given in Figure 1(a). State 0 in this simple example is the initial state (depicted with the bold circle) and state 1 is the final state (depicted with double circle). The strings *12* and *222* are among those accepted by this automaton. By using symbolic labels on this automaton in place

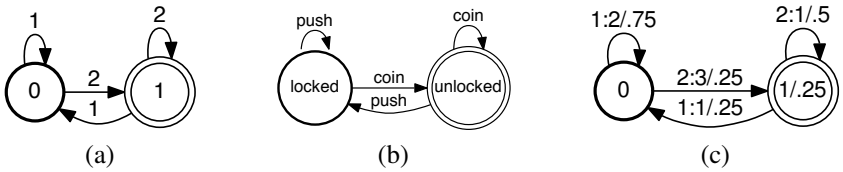

Fig. 1: (a) An example automaton; (b) a subway turnstile automaton; (c) an example weighted transducer.

of the usual integers, as depicted in Figure 1(b), we can interpret this automaton as the operation of a subway turnstile. It has two states *locked* and *unlocked* and actions (alphabet) *coin* and *push*. If the turnstile is in the locked state and you push, it remains locked and if you insert a coin, it becomes unlocked. If it is unlocked and you insert a coin it remains unlocked, but if you push once it becomes locked.

Note that directed graphs form a subset of automata with $\Sigma = \{1\}$. Thus, in what follows, we can focus on automata compression. Furthermore, to be consistent with the existing automata literature, we use states to refer to nodes and transitions to refer to edges in both graphs and automata going forward.

Some key applications motivating the study of automata are speech and natural language processing. In these applications, it is often useful to augment transitions with both an output label and some real-valued weight. The resulting devices, *weighted finite-state transducers* (FSTs), are extensively used in these fields [2], [17], [18]. An example of an FST is given in Figure 1(c). The string *12* is among those accepted by this transducer. For this input, the transducer outputs the string *23* and has weight .046875 (transitions weights 0.75 times 0.25 times final weight 0.25).

We propose an algorithm for unweighted automata compression. For FSTs, we use the same algorithm by treating the input-output label pair as a single label. If the automaton is weighted, we just add the weights at the end of the compressed file by using some standard representation.

III. Random automata compression

A. Probabilistic model

Our goal is to propose a probabilistic model for automata generation that captures real world phenomena. To this end, we first review probabilistic models for sequences and draw connections to probabilistic models for automata.

1) Probabilistic processes on sequences: We now define i.i.d. sampling of sequences. Let x_1^n denote an n-length sequence $x_1, x_2 \dots x_n$. If x_1^n are n independent samples from a distribution p over $\mathcal{X}$, then $p(x_1^n) = \prod_{i=1}^n p(x_i)$. Note that under i.i.d. sampling, the index of the sample has no importance, *i.e.,*

$$p(X_i = x) = p(X_j = x),\ \forall 1 \le i, j \le n, x \in \mathcal{X}.$$

Stationary ergodic processes generalizes i.i.d. sampling. For a stationary ergodic process p over sequences

$$p(X_i^m = x_i^m) = p(X_{i+j}^{m+j} = x_i^m), \forall i, j, m, x_i^m.$$

Informally stationary ergodic processes are those for which only the relative position of the indices matter and not the actual ones.

2) Probabilistic processes on automata: Before deriving models for automata generation, we first discuss an invariance property of automata that is useful in practice. The set of strings accepted by an automaton and the time and space of its use are not affected by the state numbering. Two automata are *isomorphic* if they coincide modulo a renumbering of the states. Thus, automata $(Q, \Sigma, \delta, q_I, F)$ and $(Q', \Sigma, \delta', q_I', F')$ are isomorphic, if there is a one-to-one mapping $f\colon Q \to Q'$ such that $f(\delta(q, a)) = \delta'(f(q), a)$, for all $q \in Q$ and $a \in \Sigma$, $f(q_I) = q_I'$, and $f(F) = F'$, where $f(F) = \{f(q)\colon q \in F\}$.

Under stationary ergodic processes, two sequences with the same order of observed symbols have the same probabilities. Similarly we wish to construct a probabilistic model of automata such that any two isomorphic automata have the same probabilities. For example, the probabilities of automata in Figure 2 are the same.

There are several probabilistic models of automata and graphs that satisfy this property. Perhaps the most studied random model is the Erdős-Rényi model $G(n, p)$, where each state is connected to every other state independently with probability p [4]. Note that if two automata are isomorphic then the Erdős-Rényi model assigns them the same probability. The Erdős-Rényi model is analogous to i.i.d. sampling on sequences. We wish to generalize the Erdős-Rényi model to more realistic models of automata.

Since the automata model should not depend on the state numbering, it could only depend on the set of incoming or leaving paths. There is no other possible semantics we could assign to states. Between incoming or outgoing paths, choosing the dependency on the incoming paths is natural given the sequentiality of language models. For example in an n-gram model, a state might have an outgoing transition with label *Francisco* or *Diego* only if it has an input transition with label *San*. This is an example where we have restrictions on paths of length 2. In general, we may have restrictions on paths of any length ℓ.

We define an *ℓ-memory* model for automata as follows. Let h_q^ℓ be the set of paths of length at most ℓ leading to the state q. The probability distribution of transitions from a state depends on the paths leading to it. Let $\delta(q, *) \stackrel{\text{def}}{=} \delta(q, 1), \delta(q, 2), \dots, \delta(q, m)$.

$$p(A) = p(\delta(1, *), \delta(2, *), \dots, \delta(n, *)) \propto \prod_{q=1}^{n} p(\delta(q, *)|h_q^\ell).$$

Similarly, transitions leaving a state q dissociate into marginals conditioned on the history h_q^ℓ and probability that $q' \in \delta(q, a)$ also dissociates into marginals.

$$\begin{aligned} p(\delta(q, *)|h_q^\ell) &= \prod_{a \in \Sigma} p(\delta(q, a)|h_q^\ell) \\ &= \prod_{a \in \Sigma} \prod_{q'=1}^{n} p(\mathbb{1}(q' \in \delta(q, a))|h_q^\ell), \end{aligned}$$

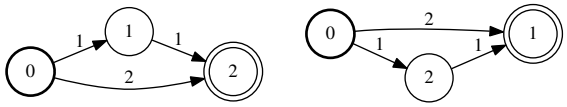

Fig. 2: An example of isomorphic automata. The above two automata are same under the permutation $0 \to 0, 1 \to 2$, and $2 \to 1$.

where $\mathbb{1}(q' \in \delta(q,a))$ is the indicator of the event $q' \in \delta(q,a)$. Note that the probabilities are defined with proportionality. This is due to the probabilities possibly not adding to one. Thus we have a constant Z to ensure that it is a probability distribution.

$$\begin{aligned} p(A) &= p(\delta(1,*), \delta(2,*), \ldots, \delta(n,*)) \\ &= \frac{1}{Z} \prod_{q=1}^{n} p(\delta(q,*)|h_q^\ell) \\ &= \frac{1}{Z} \prod_{q=1}^{n} \prod_{a \in \Sigma} p(\delta(q,a)|h_q^\ell). \end{aligned}$$

Note that *ℓ-memory* models assign the same probability to automata that are isomorphic. In our calculations, we restrict ℓ to make the model tractable.

Note that sequences form a subset of automata as follows. For a sequence x^n over alphabet Σ, consider the automata representation with states $Q = \{1, 2, \ldots n\}$, initial state $q_I = 1$, final state $F = \{n\}$, alphabet Σ, and transition function $\delta(i, x_i) = i+1$ and $\delta(i,x) = \phi$ for all $x \neq x_i$. Informally, every sequence can be represented as an automaton with line as the underlying structure. Note that the ℓ-memory model is similar to ℓ^{th} order Markov chain for sequences.

B. Entropy and coding schemes

A compression scheme is a mapping from $\mathcal{X}$ to $\{0,1\}^*$ such that the resulting code is prefix-free and can be uniquely recovered. For a coding scheme c, let $l_c(x)$ denote the length of the code for $x \in \mathcal{X}$. It is well-known that the expected number of bits used by any coding scheme is the entropy of the distribution, defined as $H(p) \stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)}$. The well known Huffman coding scheme achieves this entropy up-to one additional bit. For n-length sequences arithmetic coding is used, which achieves compression up-to entropy with few additional bits of error.

The above-mentioned coding methods such as Huffman coding and arithmetic coding require the knowledge of the underlying distribution. In many practical scenarios, the underlying distribution may be unknown and only the broader class to which the distribution belongs may be known. For example, we might know that the given n-length sequence is generated by i.i.d. sampling of some unknown distribution p over $\{1, 2, \ldots, k\}$. The objective of a universal compression scheme is to asymptotically achieve $H(p)$ bits per symbol even if the distribution is unknown. A coding scheme c for sequences over a class of distributions $\mathcal{P}$ is called universal if

$$\limsup_{n \to \infty} \max_{p \in \mathcal{P}} \frac{\mathbb{E}[l_c(X^n)] - H(X^n)}{n} = 0.$$

The normalization factor in the above definition is n, as the number of sequences of length n increases exponentially with n. For automata and graphs with n states (denoted by A_n) we choose a scaling scaling factor of n^2 as the number of automata scales as $\exp(n^2)$. We call a coding scheme c for automata over a class of distributions $\mathcal{P}$ universal if

$$\limsup_{n \to \infty} \max_{p \in \mathcal{P}} \frac{\mathbb{E}[l_c(A_n)] - H(A_n)}{n^2} = 0.$$

We now describe the algorithm LZA. Note that the algorithm does not require the knowledge of the underlying parameters or the probabilistic model.

IV. Algorithm for automaton compression

Our algorithm recursively finds substructures over states and uses a Lempel-Ziv subroutine. Our coding method is based on two auxiliary techniques to improve the compression rate: Elias-delta coding and coding the differences. We briefly discuss these techniques and their properties before describing our algorithm.

A. Elias-delta coding and coding the differences

Elias-delta coding is a universal compression scheme for integers [13]. To represent a positive integer x, Elias-delta codes use $\lfloor \log x \rfloor + 2\lfloor \log \lfloor \log x \rfloor + 1 \rfloor + 1$ bits. To obtain a code over $\mathbb{N} \cup \{0\}$, we replace x by $x+1$and use Elias-delta codes.

We now use Elias-delta codes to obtain to code sets of integers. Let $x_1, x_2, \ldots, x_m$ be integers such that $0 \leq x_1 \leq x_2 \leq \cdots \leq x_m \leq n$. We use the following algorithm to code $x_1, x_2, \ldots, x_m$. The decoding algorithm follows from ELIAS-DECODE [13].

Algorithm DIFFERENCE-ENCODE
Input: Integers $0 \leq x_1 \leq x_2 \leq \cdots \leq x_m \leq n$.
1) Use ELIAS-ENCODE to code $x_1 - 0$, $x_2 - x_1$, $\ldots x_d - x_{d-1}$.

Lemma 1 (Appendix A)**.** *For integers such that* $0 \leq x_1 \leq x_2, \ldots x_d \leq n$, DIFFERENCE-ENCODE *uses at most*

$$d \log \frac{n+d}{d} + 2d \log\Big(\log \frac{n+d}{d} + 1\Big) + d$$

bits.

We first give an example to illustrate DIFFERENCE-ENCODE's usefulness. Consider graph representation using adjacency lists. For every source state, the order in which the destination states are stored does not matter. For example, if state 1 is connected to states 2, 4, and 3, it suffices to represent the unordered set $\{2, 3, 4\}$. In general if a state is connected to d out of n states, then it suffices to encode the ordered set of states $y_1, y_2, \ldots, y_d$ where $1 \leq y_1 \leq y_2 \leq y_3 \ldots y_d \leq n$. The number of such possible sets is $\binom{n}{d}$. If the state-sets are all equally likely, then the entropy of state-sets is $\log \binom{n}{d} \approx d \log \frac{n}{d}$.

If each state is represented using $\log n$ bits, then $d \log n > d \log \frac{n}{d}$ bits are necessary, which is not optimal. However, by Lemma 1, DIFFERENCE-ENCODE uses $d \log \frac{n+d}{d}(1 + o(1)) \approx d \log \frac{n}{d}$, and hence is asymptotically optimal. Furthermore, the bounds in Lemma 1 are for the worst-case scenario and in practice DIFFERENCE-ENCODE yields much higher savings. A similar scenario arises in LZA as discussed later.

B. LZA

We now have at our disposal the tools needed to design a variant of the Lempel-Ziv algorithm for compressing automata, which we denote by LZA. Let $d_q \stackrel{\text{def}}{=} |E[q]|$ be the number of transitions from state q and let transitions in $E[q] =$

$\{(q, a_1, q_1), (q, a_2, q_2), \ldots, (q, a_{d_q}, q_{d_q})\}$ are ordered as follows: for all i, $q_i \leq q_{i+1}$ and if $q_i = q_{i+1}$ then $a_i < a_{i+1}$.

The algorithm is based on the observation that the ordering of the transitions leaving a state does not affect the definition of an automaton and works as follows. The states of the automaton are visited in a BFS order. For each state visited, the set of outgoing transitions is sorted based on their destination state. Next, the algorithm recursively finds the largest overlap of the sets of transitions that match some dictionary element and encodes the pair (matched dictionary element number, next transition), and adds the dictionary element to T_d, alphabet of the transition to T_Σ, and the destination state of the transition to T_δ. It also updates the dictionary element by adding a new dictionary element (matched dictionary element number, next transition) to the dictionary. Finally it encodes T_d, T_δ using DIFFERENCE-ENCODE and encodes each element in T_Σ using $\lceil \log m \rceil$ bits.

Algorithm LZA
Input: The transition label function δ of the automaton.
Output: Encoded sequence S.
1) Set dictionary $D = \emptyset$.
2) Visit all states q in BFS order. For every state q do:
   a) Code d_q using $\lceil \log nm \rceil$ bits.
   b) Set $T_d = \emptyset$, $T_\Sigma = \emptyset$, and $T_\delta = \emptyset$.
   c) Start with $j = 1$ in $E[q] = \{(q, a_1, q_1), \ldots, (q, a_{d_q}, q_{d_q})\}$ and continue till j reaches d_q.
      i) Find largest l such that $(a_j, q_j), \ldots, (a_{j+l}, q_{j+l}) \in D$. Let this dictionary element be d_r.
      ii) Add $(a_j, q_j), \ldots, (a_{j+l+1}, q_{j+l+1})$ to D.
      iii) Add d_r to T_d, q_{j+l+1} to T_δ, and a_{j+l+1} to T_Σ.
   d) Use DIFFERENCE-ENCODE to encode T_d, T_δ and encode each element in T_Σ using $\lceil \log m \rceil$ bits. Append these sequences to S
3) Discard the dictionary and output S.

We note that simply compressing the unordered sets T_d and T_δ suffices for unique reconstruction and thus DIFFERENCE-ENCODE is the natural choice. Observe that DIFFERENCE-ENCODE is a succinct representation of the dictionary and does not affect the way Lempel-Ziv dictionary is built. Thus the decoding algorithm follows immediately from retracing the steps in LZA and LZ78 decoding algorithm. The run time of LZA is similar to that of other Lempel Ziv algorithms and is $\mathcal{O}(n+|E|)$.

If DIFFERENCE-ENCODE is not used, the number of bits used would be approximately $|D| \log |D| + |D| \log n$, which is strictly greater than that number of bits in Lemma 2. Furthermore, we did consider several other natural variants of this algorithm where we difference-encode the states first and then serialize the data using a standard Lempel-Ziv algorithm. However, we could not prove asymptotic optimality for those variants. Proving their non-optimality requires constructing distributions for which the algorithm is non-optimal and is not the focus of this paper.

We first bound the number of bits used by LZA in terms of the size of the dictionary $|D|$, the number of states n, and the alphabet size m. This bound is independent of the underlying probabilistic model. Next, we proceed to derive probabilistic bounds.

Lemma 2 (Appendix B)**.** *The total number of bits used by* LZA *is at most*

$$|D|\left[\log(n+1) + \log(\nu+1) + 2\log(\log(n+1)+1)\right] + |D|\left[2\log(\log(\nu+1)+1) + 2 + \lceil \log m \rceil\right] + n\lceil \log nm \rceil,$$

where $\nu = \frac{n^2}{|D|}$.

C. Proof of optimality

In this section, we prove that LZA is asymptotically optimal for the random automata model introduced in Section III. Lemma 2 gives an upper bound on the number of bits used in terms of the size of the dictionary $|D|$. We now present a lower bound on the entropy in terms of D which will help us prove this result. The proof is given in Appendix C.

Lemma 3. LZA *satisfies*

$$H(p) \geq \mathbb{E}[|D|]\left[\log \frac{\mathbb{E}[|D|]}{n} - m^\ell - \log\left(\frac{n^2 m}{\mathbb{E}[|D|]} + 1\right) - 1\right].$$

The above result together with Lemma 2 implies

Theorem 4 (Appendix D)**.** *If* $2^{m^\ell} = o\left(\frac{\log n}{\log\log n}\right)$*, then* LZA *is a universal compression algorithm.*

V. EXPERIMENTS

A. Automaton structure compression

LZA compresses automata, but for most applications, it is sufficient to compress the automata structure. We convert LZA into LZA_S, an algorithm for automata structure compression as follows. We first perform a breadth first search (BFS) with the initial state as the root state and relabel the states in their BFS visitation order. We then run LZA with the following modification. In step 2, for every state q we divide the transitions from q into two groups, T^q_{old} transitions whose destination states have been traversed before in LZA and T^q_{new}, transitions whose destinations have not been traversed. Note that since the state numbers are ordered based on a BFS visit, the destination state numbers in T^q_{new} are $1, 2, \ldots n$, and can be recovered easily while decoding and thus need not be stored. Hence, we run step $2b$ in LZA only on transitions in T^q_{old}. For T^q_{new}, we just compress the transition labels using a standard implementation of LZ78.

Since each destination state can appear in T^q_{new} only once, the number of transitions in $\cup_q T^q_{\text{new}} \leq n$, Since this number is $\ll n^2$, the normalization factor in the definition of universal compression algorithm for automata, the proof of Theorem 4 extends to LZA_s. Since for most applications, it is sufficient to compress to the automata structure, we implemented LZA_s in C++ and added it to the *OpenFst* open-source library [3][1].

B. Comparison

The best known convergence rate of all Lempel-Ziv algorithms for sequences is $\mathcal{O}\left(\frac{\log\log n}{\log n}\right)$ and LZA has the same convergence rate under the ℓ-memory probabilistic model.

However in practice data sets have finitely many states and the underlying automata may not be generated from an ℓ-memory probabilistic model. To prove the practicality of the algorithm, we compare LZA_s with the Unix compress command (LZW) and gzip (LZ77 and Huffman coding) for various synthetic and real data sets.

[1] Available for download atwww.openfst.org

Class	Uncompressed	LZA_S	compress	gzip	LZA+gzip
G_1	172208	18260	22681	23752	17320
A_1	172076	21745	33478	31682	21108
G_2	34102	2536	4994	4564	2443
A_2	34171	3027	6707	5546	2940

TABLE I: Synthetic data compression examples (in bytes).

1) Synthetic Data: While the ℓ-memory probabilistic model illustrates a broad class of probabilistic models on which LZA is universal, generating samples from an ℓ-memory model is difficult similar to graphical models as the normalization factor Z is hard to compute. We therefore test our algorithm on a few simpler synthetic data sets. In all our experiments the number of states is 1000 and the results are averaged over 1000 runs.

Table I summarizes our results for a few synthetic data sets, specified in bytes. Note that one of the main advantages of LZA_S over existing algorithms is that LZA_S just compresses the structure, which is sufficient for applications in speech processing and language modeling. Furthermore, note that to obtain the actual automaton from the structure we need the original state numbering, which can be specified in $n \log n$ bits, which is $(1000 \log_2 1000)/8 \leq 1250$ bytes in our experiments. Even if we add 1250 bytes to our results in Table I, LZA still performs better than gzip and compress.

We run the algorithm on four different synthetic data sets G_1, G_2, A_1, A_2. G_1 and A_1 are models with a uniform out-degree distribution over the states and G_2 and A_2 are models with a non-uniform out-degree distribution:

G_1: directed Erdős-Rényi graphs where we randomly generate transitions between every source-destination pair with probability $1/100$.

A_1: automata version of Erdős-Rényi graphs, where there is a transition between every two states with probability $1/100$ and the transition labels are chosen independently from an alphabet of size 10 for each transition.

G_2: we assign each state a class $c \in \{1, 2, \ldots 1000\}$ randomly. We connect every two states s and d with probability $1/(c_s + c_d)$. The resulting graph has degrees varying from 2 to $\log 1000$.

A_2: we generate the transitions as above and we label each transition to be a deterministic function of the destination state. This is similar to n-gram models, where the the destination state determines the label. Here again we chose $|\Sigma| = 10$.

Note that LZA always performs better than the standard Lempel-Ziv-based algorithms gzip and compress. Note that algorithms designed with specific knowledge of the underlying model can achieve better performance. For example, for G_1, arithmetic coding can be used to obtain a compressed file size of $n^2 h(0.01)/8 \approx 10000$ bytes. However the same algorithm would not perform well for G_2 or A_2.

2) Real-World Data: We also tested our compression algorithm on a variety of 'real-world' automata drawn from various speech and natural language applications. These include large speech recognition language models and decoder graphs [17], text normalization grammars for text-to-speech [19], speech recognition and machine translation lattices [16], and pair n-gram grapheme-to-phoneme models [7]. We selected approximately eighty such automata from these tasks and removed their weights and output labels (if any), since we focus here on unweighted automata. Figure 3 shows the compressed sizes of these automata,

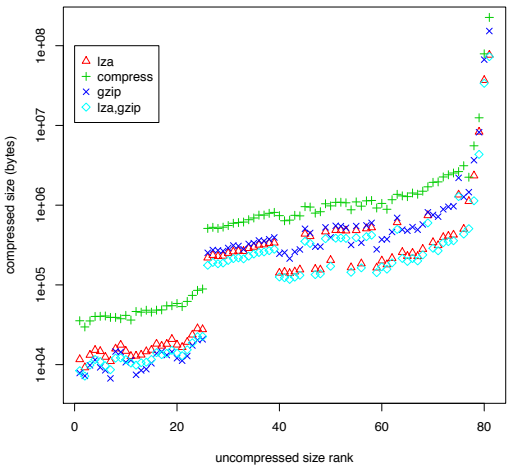

Fig. 3: Real-world compression examples.

ordered by their uncompressed (adjacency-list) size rank, with the same set of compression algorithms presented in the synthetic case. At the smallest sizes, gzip out-performs LZA, but after about 100 kbytes in compressed size, LZA is better. Overall, the combination of LZA and gzip performs best.

VI. Acknowledgements

We thank J. Acharya and A. Orlitsky for helpful discussions.

References

[1] M. Adler and M. Mitzenmacher. Towards compressing web graphs. In *DCC*, 2001.
[2] V. Alabau, F. Casacuberta, E. Vidal, and A. Juan. Inference of stochastic finite-state transducers using *N* -gram mixtures. In *IbPRIA*, 2007.
[3] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. Openfst: A general and efficient weighted finite-state transducer library. In *CIAA*, 2007.
[4] N. Alon and J. Spencer. *The Probabilistic Method*. Wiley, 1992.
[5] V. N. Anh and A. Moffat. Local modeling for webgraph compression. In *DCC*, 2010.
[6] A. Apostolico and G. Drovandi. Graph compression by BFS. *Algorithms*, 2(3):1031–1044, 2009.
[7] M. Bisani and H. Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, 2008.
[8] Y. Choi and W. Szpankowski. Compression of graphical structures. In *ISIT*, 2009.
[9] Y. Choi and W. Szpankowski. Compression of graphical structures: Fundamental limits, algorithms, and experiments. *IEEE Trans. on Info. Theory*, 58(2):620–638, 2012.
[10] T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley, 2006.
[11] J. Daciuk. Experiments with automata compression. In *CIAA*, 2000.
[12] J. Daciuk and J. Piskorski. Gazetteer compression technique based on substructure recognition. In *IIPWM*. Springer, 2006.
[13] P. Elias. Universal codeword sets and representations of the integers. *IEEE Tran. on Info. Theory*, 21(2):194–203, 1975.
[14] S. Grabowski and W. Bieniecki. Tight and simple web graph compression. In *Proceedings of the Prague Stringology Conference*, 2010.
[15] G. Hansel, D. Perrin, and I. Simon. Compression and entropy. In *STACS*, 1992.
[16] G. Iglesias, C. Allauzen, W. Byrne, A. de Gispert, and M. Riley. Hierarchical phrase-based translation representations. In *Proc. of Conf. on Empirical Methods in Natural Language Processing*, 2011.
[17] M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1), 2002.
[18] E. Roche and Y. Schabes. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 1995.
[19] T. Tai, W. Skut, and R. Sproat. Thrax: An open source grammar compiler built on openfst. In *ASRU*, 2011.
[20] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. on Info. Theory*, 23(3):337–343, 1977.

APPENDIX

A. Proof of Lemma 1

Since 0 is included in the set, the number of bits used to represent x is upper bounded by $\theta(x) = \log(x+1)+2\log(\log(x+1)+1)\rfloor + 1$. Observe that θ is a concave function since both $\log$ and $x \mapsto \log(\log x)$ are concave. Let $x_0 = 0$. Then, by the concavity of θ, the total number of bits B used can be bounded as follows:

$$\begin{aligned} B &\leq \sum_{i=1}^{d} \theta(x_i - x_{i-1}) \\ &= d \sum_{i=1}^{d} \frac{1}{d}\theta(x_i - x_{i-1}) \\ &\leq d\,\theta\Big(\frac{1}{d}\sum_{i=1}^{d} x_i - x_{i-1}\Big) = d\,\theta\Big(\frac{1}{d}x_n\Big) \leq d\,\theta\Big(\frac{n}{d}\Big), \end{aligned}$$

where we used for the last inequality $x_n \leq n$ and the fact that θ is an increasing function. This completes the proof of the lemma.

B. Proof of Lemma 2

Let k_q be the number of elements added to the dictionary when state q is visited by LZA. The maximum value of the destination state is n. Thus, by Lemma 1, the number of bits used to code T_δ is at most

$$\sum_{q=1}^{n} \Big(k_q \theta\Big(\frac{n}{k_q}\Big) + k_q \lceil \log m \rceil\Big),$$

where $\theta(\cdot)$ is the function introduced in the proof of Lemma 1. Similarly, since the maximum value of any dictionary element is $|D|$, by Lemma 1, the number of bits used to code T_d is at most

$$\sum_{q=1}^{n} k_q \theta\Big(\frac{|D|}{k_q}\Big).$$

By concavity these summations are maximized when $k_q = \frac{|D|}{n}$ for all q. Plugging in that expression in the sums above yields the following upper bound on the maximum number of bits used:

$$|D|\theta(\nu) + |D|\lceil \log m \rceil + |D|\theta(n).$$

Additionally, this number must be augmented by $n\lceil \log nm \rceil$ since $\lceil \log nm \rceil$ bits are used to encode each d_q, which completes the proof.

C. Proof of Lemma 3

One of the main technical tools we use is Ziv's inequality, which is stated below.

Lemma 5 (Variation of Ziv's inequality)**.** *For a probability distribution p over non-negative integers with mean μ,*

$$H(p) \leq \log(\mu + 1) + 1.$$

The next lemma bounds the probability of disjoint events under different distributions.

Lemma 6. *If $A_1, A_2, \ldots, A_k$ be a set of disjoint events. Then for a set of distributions $p_1, p_2, \ldots p_r$,*

$$\sum_{i=1}^{k}\sum_{j=1}^{r} p_j(A_k) \leq \sum_{j=1}^{r} 1 = r.$$

We now lower bound $H(p)$ in terms of the number of dictionary elements.

Let $d_q \stackrel{\text{def}}{=} |E[q]|$ be the number of transitions from state q and let transitions in $E[q] = \{(q, a_1, q_1), \ldots, (q, a_{d_q}, q_{d_q})\}$ are ordered as follows: for all i, $q_i \leq q_{i+1}$ and if $q_i = q_{i+1}$ then $a_i < a_{i+1}$. To simplify the discussion, we will use the shorthand $e_{q,i} \stackrel{\text{def}}{=} (q, a_i, q_i)$. Then, by the definition of our probabilistic model,

$$\log p(A) = \sum_{q=1}^{n} \log p(e_{q,1}, e_{q,2}, \ldots, e_{q,d_q} | h_q^\ell) - \log Z.$$

We group the transitions the way LZA constructed the dictionary. Let $\{D_{q,i}\}$ be the set of dictionary elements added when state q is visited during the execution of the algorithm. For a dictionary element $D_{q,i}$, let $s_{q,q'}$ be the starting $e_{q,i}$ and $t_{q,q'}$ the terminal $e_{q,i}$. Then, by the independence of the transition labels and the fact that $Z \geq 1$,

$$\log p(A) \leq \sum_{q=1}^{n}\sum_{D_{q,i}} \log p(e_{q,s_{q,i}}, e_{q,s_{q,i}+1}, \ldots e_{q,t_{q,i}} | h_q^\ell).$$

Let $g_{q,i} = t_{q,i} - s_{q,i}$. We group them now with $g_{q,i}$ and $s_{q,i}$. Let $\mathcal{D}(s,g)$ be the set of dictionary elements $D_{q,i}$ with $s_{q,i} = s$ and $g_{q,i} = g$ and let $c_{s,g}$ be the cardinality of that set: $c_{s,g} = |\mathcal{D}(s,g)|$ and $e_{q,s}^t = e_{q,s}, e_{q,s+1}, \ldots e_{q,t}$. Then, by Jensen's inequality, we can write

$$\begin{aligned} &\sum_{q=1}^{n}\sum_{D_{q,i}} \log p(e_{q,s_{q,i}}^{t_{q,i}} | h_q^\ell) \\ &= \sum_{q=1}^{n}\sum_{s=1}^{n}\sum_{g=1}^{n}\sum_{D_{q,i}\in\mathcal{D}(s,g)} \log p(e_{q,s}^{s+g} | h_q^\ell) \\ &= \sum_{s=1}^{n}\sum_{g=1}^{n} c_{s,g} \frac{1}{c_{s,g}} \sum_{q=1}^{n}\sum_{D_{q,i}\in\mathcal{D}(s,g)} \log p(e_{q,s}^{s+g} | h_q^\ell) \\ &\leq \sum_{s=1}^{n}\sum_{g=1}^{n} c_{s,g} \log \frac{1}{c_{s,g}} \sum_{q=1}^{n}\sum_{D_{q,i}\in\mathcal{D}(s,g)} p(e_{q,s}^{s+g} | h_q^\ell) \\ &\leq \sum_{s=1}^{n}\sum_{g=1}^{n} c_{s,g} \log \frac{2^{m^\ell}}{c_{s,g}}. \end{aligned}$$

where the last inequality follows by Lemma 6, the fact that the events in each summation are disjoint and mutually exclusive and that the number of possible histories h_q^ℓ is $\leq 2^{m^l}$. Since $\sum_{s,g} c_{s,g} = |D|$,

$$\begin{aligned} &\sum_{s=1}^{n}\sum_{g=1}^{n} c_{s,g} \log \frac{2^{m^\ell}}{c_{s,g}} \\ &= |D|m^\ell + \sum_{s=1}^{n}\sum_{g=1}^{n} c_{s,g} \log \frac{1}{c_{s,g}} \\ &= |D|m^\ell - |D|\log|D| + |D| \sum_{s=1}^{n}\sum_{g=1}^{n} \frac{c_{s,g}}{D} \log \frac{|D|}{c_{s,g}} \\ &= |D|m^\ell - |D|\log|D| + |D|H(c_{s,g}). \end{aligned}$$

Let c_s and c_g be the projections of $c_{s,g}$ into first and second coordinates. Then, we can write

$$H(c_{s,g}) \leq H(c_s) + H(c_g) \leq \log n + H(c_g).$$

Using $\sum_{s,g} c_{s,g} g \leq n^2 m$, by Ziv's inequality, the following holds: $H(c_g) \leq \log\left(\frac{n^2 m}{|D|}+1\right)$. Combining this with the previous inequalities gives

$$\log p(A) \leq |D| \left[m^\ell + \log\left(\frac{n^2 m}{|D|}+1\right) + 1 - \log\frac{|D|}{n} \right].$$

Taking the expectation of both sides, next using the concavity of $|D| \mapsto -|D|\log(|D|)$ and Jensen's inequality yield

$$H(p) \geq \mathbb{E}[|D|] \left[\log\frac{\mathbb{E}[|D|]}{n} - m^\ell - \log\left(\frac{n^2 m}{\mathbb{E}[|D|]}+1\right) - 1 \right].$$

D. Proof of Theorem 4

We first upper bound $\mathbb{E}[|D|]$ using Lemma 3.

Lemma 7. *For the dictionary D generated by* LZA

$$\mathbb{E}[|D|] \leq \frac{10 n^2 m \log(m+1) 2^{m^\ell}}{\log n}.$$

Proof: An automaton is a random variable over n^2 transition labels each taking at most $m^m + 1$ values, hence $H(p) \leq n^2 m \log(m+1)$. Combining this inequality with Lemma 3 yields

$$\begin{aligned} &n^2 m \log(m+1) \geq \\ &\mathbb{E}[|D|] \left[\log\frac{\mathbb{E}[|D|]}{n} - m^\ell - \log\left(\frac{n^2 m}{\mathbb{E}[|D|]}+1\right) - 1 \right]. \end{aligned}$$

Now, let $U = \frac{10 n^2 m \log(m+1) 2^{m^\ell}}{\log n}$ and assume that the inequality $\mathbb{E}[|D|] > U$ holds. Then, the following inequalities hold:

$$\begin{aligned} &\mathbb{E}[|D|] \left[\log\frac{\mathbb{E}[|D|]}{n} - m^\ell - \log\left(\frac{n^2 m}{\mathbb{E}[|D|]}+1\right) - 1 \right] \\ &> U \left[\log\frac{10 m n \log(m+1) 2^{m^\ell}}{\log n} - m^\ell \right] \\ &+ U \left[-\log\left(\frac{\log n}{102^{m^\ell}\log(m+1)}+1\right) - 1 \right] \\ &\geq U \left[\log n - \log\left(\frac{\log n}{102^{m^\ell}\log(m+1)}+1\right) \right] \\ &- U \left[\log\log n\right] \\ &> n^2 m \log(m+1), \end{aligned}$$

which leads to a contradiction. This completes the proof of the lemma. ∎

We now have all the tools to prove Theorem 4. Let $W(|D|)$ be the upper bound in Lemma 2. Since we have a probabilistic model and the fact that W is concave in $|D|$, the expected number of bits

$$\mathbb{E}[l_{LDA}(A_n)] \leq W(\mathbb{E}[|D|]).$$

Substituting the lower bound on $H(p)$ from Lemma 3 and rearranging terms, we have

$$\begin{aligned} &\max_{p(A_n)} \frac{\mathbb{E}[l_{\mathrm{LZA}}(a_n)] - H(p)}{n^2} \\ &= \max_{p(A_n)} \frac{W(\mathbb{E}[|D|]) - H(p)}{n^2} \\ &\leq \frac{\mathbb{E}[|D|]}{n^2} \left[\log\left(\frac{(n+1)n}{\mathbb{E}[|D|]} \cdot \left(\frac{n^2 m}{\mathbb{E}[|D|]} + 1 \right)^2 \right) \right] \\ &+ \frac{\mathbb{E}[|D|]}{n^2} \left[2\log\left(\log\left(\frac{n^2}{\mathbb{E}[|D|]} + 1 \right) + 1 \right) + m^\ell + 4 + \log m \right] \\ &\frac{2\mathbb{E}[|D|]}{n^2} \log(\log(n+1)+1) + \frac{\lceil \log nm \rceil}{n} \\ &= \mathcal{O}\left(2^{m^\ell} \frac{\log\log n + m^\ell}{\log n} \right). \end{aligned}$$

The last equality follows from Lemma 7. As $n \to \infty$, the bound goes to 0 and hence LZA is a universal compression algorithm.