Universal Guessing with Side Information for Binary Sequence Estimation

Keren Mazaki and Raz Yehuda
Advisor: Prof. Asaf Cohen

***Abstract*—This study explores the application of universal guessing techniques to sequence estimation, given side information. We implement and analyze three variants of Lempel-Ziv (LZ) compression-based algorithms, building upon the universal guessing algorithm proposed by A. Cohen and N. Merhav in [1]: a baseline method using only side information, and two methods that learn from a known prefix of the sequence, in addition to side information. We impalement and analyze of the LZ compression algorithm, both with and without side information. We then apply these concepts to universal guessing algorithms for sequence estimation. We evaluate the performance of these algorithms on binary sequences, under different noise levels and sequence complexities. We also demonstrate the practical application of these algorithms in binary image reconstruction tasks, comparing their performance in denoising under different noise conditions.**

I. Introduction

Estimating an unknown sequence from a noisy observation is a fundamental problem in information theory, communications, and cryptography. This work explores the application of universal guessing techniques, to this estimation task. The universal guessing framework provides a theoretical foundation for designing estimators that can perform well across a wide range of unknown source distributions.

Our study focuses on three main components:

1) Implementation and analysis of the Lempel-Ziv (LZ) compression algorithm, both with and without side information.
2) Application of a universal guessing algorithm based on LZ decoding with side information for a sequence estimation,proposed in [1].
3) Development of improved algorithms combining LZ decoding of a known prefix and guessing to improve estimation accuracy.

Through these investigations, we aim to provide insights into the practical application of universal guessing techniques and their limitations in real-world sequence estimation tasks.

II. Lempel-Ziv Compression

The Lempel-Ziv (LZ) algorithm [2] is a fundamental lossless data compression technique that forms the basis for many practical compression systems. In this study, we implemented two variants of the LZ algorithm:

1) Standard LZ compression.
2) LZ compression with side information, where side information refers to an additional sequence correlated with the primary sequence to be compressed. This side information is known to both the encoder and decoder and can be used to improve compression efficiency.

A. LZ Algorithm Without Side Information

In this section, we describe the implementation of the standard LZ algorithm on a sequence x.

a) Encoding: The encoding process parses the input sequence x into unique phrases:

Algorithm 1 LZ Compression Without Side Information - Encoding

Let α be the size of the alphabet (number of distinct symbols)
Initialize an empty dictionary D
Initialize an empty list I for the encoded results
while not end of input sequence x **do**
    Find longest prefix p in D matching current position in x
    Let s be the next symbol after p in x
    $\pi \leftarrow$ index of p in D (0 if no prefix was found)
    $I_a \leftarrow s$
    add $\alpha \cdot \pi + I_a$ to I
    Add p concatenated to s to D
end while
Output I

b) Decoding: The decoding process reconstructs the original sequence from the encoded data:

Algorithm 2 LZ Compression Without Side Information - Decoding

Let α be the size of the alphabet
Initialize empty dictionary D
for each encoded index in list I **do**
    $I_a \leftarrow I \bmod \alpha$
    $\pi \leftarrow \frac{I - I_a}{\alpha}$
    if $\pi = 0$ **then**
        add I_a to D
    else
        Look up phrase p corresponding to π in D
        add p concatenated to I_a to D
    end if
end for
output D

B. Compression Rate Analysis For Basic LZ compression

We analyzed the compression rate, defined as:

$$u(x) = \frac{1}{n} C(x) \log C(x)$$

where $C(x)$ is the number of phrases in the decoded data. We implement the compression algorithm on different stochastic binary sequences. From [3], theorom 13.5.1, we know that:

$$\lim_{n \to \infty} u(x) = H(X)$$

where $H(x)$ is the entropy rate of the sequence.

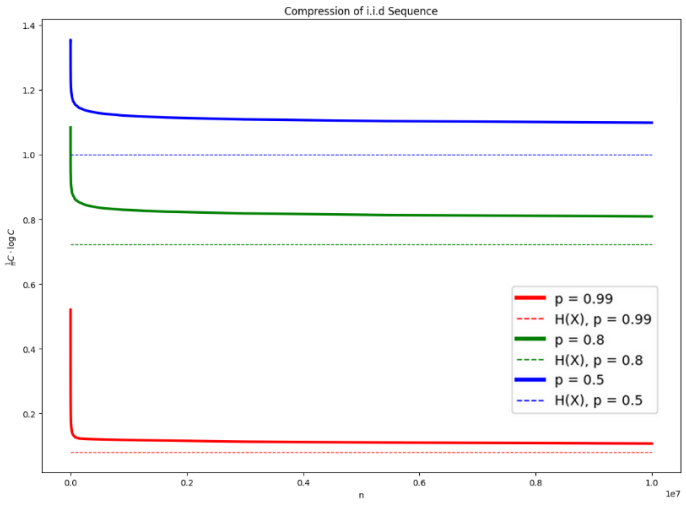

Fig. 1. Compression rate of different i.i.d sequences with probabilities $x \sim \text{Ber}(p)$

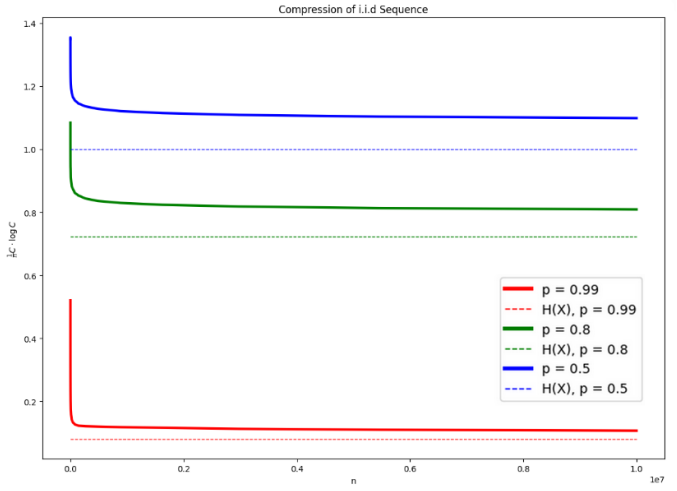

Fig. 2. Compression rate of different first order Markov chains. The transition matrix for each sequence is given by $\begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix}$, and a stationary probability $x_0 \sim \text{Ber}(\frac{1}{2})$.

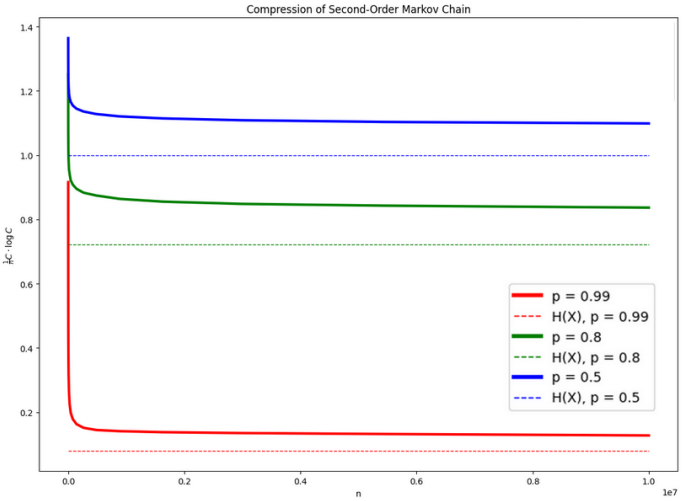

Fig. 3. Compression rate of different second order Markov chains. The transition matrix for each sequence is given by $\begin{pmatrix} p & 1-p \\ 1-p & p \\ 1-p & p \\ p & 1-p \end{pmatrix}$, and a stationary probability $x_{-1}, x_0 \sim \text{Ber}(\frac{1}{2})$.

Figures 1, 2, and 3 show the compression rates for i.i.d. sequences, first-order Markov chains, and second-order Markov chains, respectively, with binary alphabet. The analysis of these graphs reveals several key observations:

1) **Compression rate convergence:**
   - for all cases, we can see that the compression rate converges to the entropy rate, as expected.
2) **Compression rate vs. probability:**
   - The rate of convergence varies depending on the randomness of the sequence:

- For highly biased sequences (e.g., $p = 0.99$), convergence to the entropy rate is relatively quick.
- For more random sequences (e.g., $p = 0.5$), convergence is slower and the compression rate remains above the entropy rate for longer sequence lengths.

3) **Comparison across models:**

- The i.i.d. sequence (Figure 1) shows the best convergence in the different probability curves.
- The first-order Markov chain (Figure 2) shows slightly slower convergence of the curves, indicating that the sequence is less trivial.
- The second-order Markov chain (Figure 3) shows an even slower convergence of the curves.

C. LZ Algorithm With Side Information

In scenarios where an additional correlated sequence y is available, the LZ algorithm can be adapted to use this side information to improve compression efficiency.

a) Encoding: The encoding process codes the input sequence x, using a correlated sequence y:

Algorithm 3 LZ Compression With Side Information - Encoding

Let α be the size of the alphabet
Initialize an empty dictionary D_x for each unique y phrase
Initialize empty list L for phrase lengths
Initialize and empty list I for the encoded results
while not end of input sequences x and y **do**
    Find longest prefix pair (x_i, y_i) where x_i is in D_x of y_i
    Let s be the next symbol after x_i in x
    $\pi \leftarrow$ index of x_i in D_x of y_i (0 if no prefix was found)
    $I_a \leftarrow s$
    add $\alpha \cdot \pi + I_a$ to I
    add x_i concatenated to I_a to D_x of y_i
    Add $length(x_i) + 1$ to L
end while
Output L and I.

b) Decoding: The decoding process reconstructs the original sequence x using the encoded data, side information y, and the lengths list of the phrases:

Algorithm 4 LZ Compression With Side Information - Decoding

Let α be the size of the alphabet
Parse y according to lengths list L
Initialize an empty dictionary D_x for each unique y phrase
for each encoded index I and corresponding y phrase **do**
    $I_a \leftarrow I \bmod \alpha$
    $\pi \leftarrow \frac{I - I_a}{\alpha}$
    if $\pi = 0$ **then**
        add I_a to D_x of y_i
    else
        Look up phrase p corresponding to π in D_x of y_i
        add p concatenated to I_a to D_x of y_i
    end if
    output decoded phrases of x by order
end for

D. Compression Rate Analysis For Side Information LZ

For compression with side information, the compression rate in defined in [1] as:

$$u(x|y) = \frac{1}{n} \sum_{i=1}^{c(y)} c_i(x|y) \log c_i(x|y)$$

where $c(y)$ is the number of distinct y-phrases in the joint parsing, and $c_i(x|y)$ is the number of distinct x-phrases that appear jointly with the i-th distinct y-phrase.

From [4], we also know that:

$$\lim_{n \to \infty} u(x|y) = H(X|Y)$$

where $H(x|y)$ is the conditioned entropy rate of x given y.

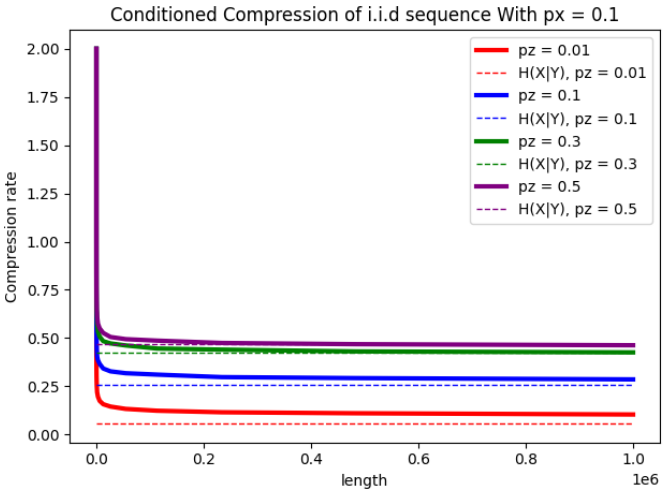

Fig. 4. Conditioned Compression rate of i.i.d sequences with probability $x \sim \text{Ber}(0.1)$, given side information $y = x \oplus z$, where z is an i.i.d noise sequence with probabilities $z \sim \text{Ber}(p_z)$

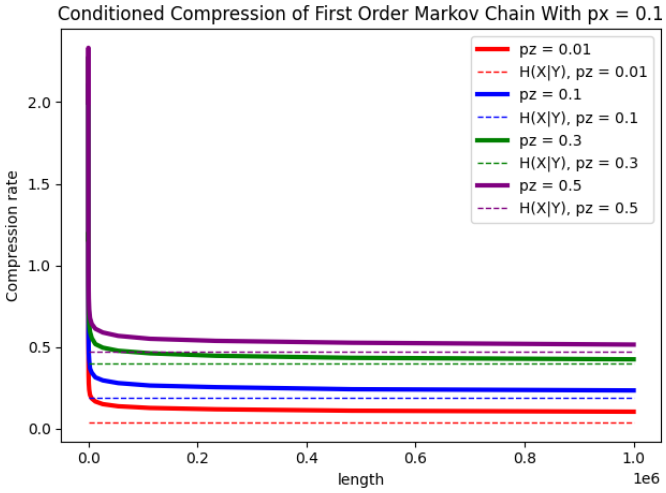

Fig. 5. Conditioned Compression rate of first order Markov chains with transition matrix $\begin{pmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{pmatrix}$, and a stationary probability $x_0 \sim \text{Ber}(\frac{1}{2})$, given side information $y = x \oplus z$, where z is an i.i.d noise sequence with probabilities $z \sim \text{Ber}(p_z)$

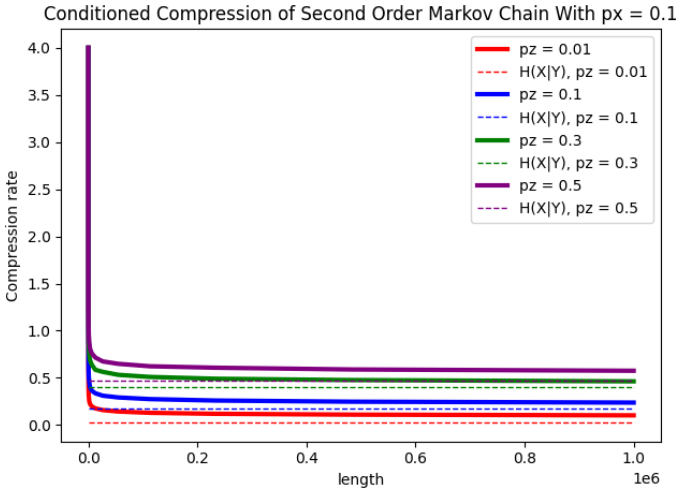

Fig. 6. Conditioned Compression rate of second order Markov chains with transition matrix $\begin{pmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \\ 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$, and a stationary probability $x_{-1}, x_0 \sim \text{Ber}(\frac{1}{2})$, given side information $y = x \oplus z$, where z is an i.i.d noise sequence with probabilities $z \sim \text{Ber}(p_z)$

Figures 4, 5, and 6 show the conditioned compression rates for i.i.d. sequences, first-order Markov chains, and second-order Markov chains, respectively, with binary alphabet, given side information y.

For figures 5, and 6, notice that y is a hidden Markov chain, and calculating the entropy rate $H(X|Y)$ is very complex. We used an approximation, $\hat{H}(X|Y)$, following the next inequality:

$$H(X|Y) = H(x_i|x_{i-1}, y_0^n) \leq H(H(x_i|x_{i-1}, y_i, y_{i+1}) = \hat{H}(X|Y) \tag{1}$$

This inequality holds since conditioning reduces entropy.

The analysis of these graphs reveals several key observations:

1) **Convergence:**
   - The compression rates do converge to $H(X|Y)$, matching the theoretical limit.
   - The compression rates converge relatively quickly as the sequence length increases.
2) **Influence of p_z (correlation between x and y):**
   - Lower p_z values (e.g., 0.01) result in better compression rates, as they indicate stronger correlation between x and y. in these cases, since $H(X|Y) \approx 0$, the compression rate approaches 0.
   - Higher p_z values (e.g., 0.5) lead to compression rates closer to those without side information, as x and y become independent, and we get $H(X|Y) = H(X)$.
3) **Effect of side information:**
   - The compression rates with side information are generally lower than without side information, indicating improved compression efficiency.
   - The degree of improvement varies depending on the correlation between x and y, as represented by different p_z values.

III. Implementation of Universal Guessing Algorithms with Side Information

We implemented and extended three variants of the Lempel-Ziv (LZ) decoder-based algorithm for universal guessing with side information, as proposed in [1]. Our work addresses practical challenges in implementation and introduces modifications to enhance performance by providing more information about the sequence x. We present three variants of the guessing algorithm. These three variants represent a progression in the utilization of available information:

- A_1 uses only side information y.
- A_2 leverages a known prefix of x to improve guessing by learning.
- A_3 assumes we can learn best by only basing on the known prefix of x, then utilizing all observed patterns in the known part of x.

Each variant builds upon the previous, aiming to improve guessing efficiency.

A. Algorithm A_1: Guessing with Side Information Only

This baseline algorithm implements the core concept of universal guessing using only the side information y to generate guesses for x. It operates without any prior knowledge of the relationship between x and y.

Algorithm A1 Universal Guessing with Side Information Only

```
Let α be the size of the alphabet
Initialize empty dictionaries D_x, D_I and D_L for each unique y phrase
end ← 0
while not end of y do
    beg ← end
    L ← Generate_L(min_L, max_L, D_x, D_L)
    end ← beg + L
    phrase ← y[beg : end]
    I ← Generate_I(D_I, D_x)
    I_a ← I mod α
    π ← (I − I_a)/α
    if π = 0 then
        add I_a to D_x
    else
        Look up phrase p corresponding to π in D_x
        add p concatenated to I_a to D_x
        output p concatenated to I_a
    end if
    add L and I to D_I, D_L respectively
    min_L, max_L ← UpdateBounds(L, D_L)
end while
```

1) Function Descriptions:

- **Generate_L(min_L, max_L, D_x, D_L)**: This function randomly chooses L between min_L and max_L, and then ensures that the chosen L satisfies the validation criteria (explained in detail in III-A2).
- **Generate_I(D_I, D_x)**: This function randomly chooses I between 0 and $\alpha(\cdot length(D_x)+1)$, that is α times the number of past phrases of x under the prefix of the current phrase of y, plus 1. This range applies since $I = \alpha \cdot \pi + I_A$, where $0 \le \pi \le length(D_x)$, and $0 \le I_A \le \alpha$. The function then ensures that the chosen I satisfies the validation criteria (explained in detail in III-A2).
- **UpdateBounds(L, D_L)**: This function updates the minimum and maximum bounds for L (also explained in detail in III-A2).

2) Key Challenges and Solutions: When implementing algorithm A_1 based on the guessing algorithm described in [1], we found a few problems that required changes in order to maintain a valid parsing of x and y that could have been produced by the LZ decoding algorithm.

- Phrase Length Validation: A critical challenge in implementing these algorithms was determining valid values for the phrase length L. A length L is not valid if:
  - The current phrase's prefix has not been seen before. For example: y phrase = 110 but the phrase 11 isn't in the past of y.
  - The current phrase of y has already received all possible values for phrases of x in the past. For example: y phrase = 11, and in the past of that phrase we have x phrase = 00, 01, 10, 11.
  - the current phrase of y has appeared more than twice the number of times its prefix has appeared. For example: current y phrase is 11. In the past we got the phrase's prefix, 1 with x-phrase = 1, and the y phrase 11 already appeared twice with the x-phrases 11 and 10. There aren't any new options for phrases of x in this case.
- Bound Adjustment of Phrase Length: In order to randomly select the length L for each phrase, we implemented a mechanism to dynamically update the bounds of L:
  - $\text{min\_L} = \max\{L | N_L < 4^L\}$, where N_L is the number of times L has appeared in the past. This means that if a certain length L has appeared more than 4^L times, the next phrase length must be larger, since all of the options for a phrase pair (x_i, y_i) with length L already appeared in the past.
  - $\text{max\_L} = \max\{L | L \text{ was in the past}\} + 1$, that means that a certain length L can appear only if $L-1$ already appeared at least once.
- Phrase Index Validation: Another challenge we faced was making sure that the index I is valid. I is not valid if:
  - $L > 1$ and $I = 0$ or $I = 1$. In this case, the value of I indicates that there is no prefix for the current y phrase, and that is only possible for phrases that are with a length equal to 1.
  - I appeared in the past of the current y phrase. Since I corresponds to a specific prefix of the x phrase, and the new symbol, if the same I appears more than once under the same phrase of y, it means that the joint phrase of (x, y) repeats itself, and that contradic ts the concept of the LZ algorithm.
- LZ Algorithm Correction: In the original paper [1], the authors proposed a method for guessing (without side information) that involves feeding purely random bits into the LZ decoder in order to produce a random I for each phrase guess, until a sequence of length n is obtained. The probability of obtaining a sequence x at the decoder output was originally given by the equation:

$$\tilde{P}(x) = \prod_{j=1}^{c(x)+1} 2^{-\hat{L}_j} \tag{2}$$

where $c(x)$ is the number of phrases in the LZ parsing of x, and $\hat{L}_j$ is the length of the codeword for the j-th phrase.

The purpose of this equation was to show that the probability of generating any sequence x using this method is of the exponential order of $2^{-c(x)\log c(x)}$.

However, since our implementation addressed the problem of generating repeated phrases by disqualifying an index I if it appeared in the past, I's distribution has changed. This change affects the original equation. The equality becomes an inequality:

$$\tilde{P}(x) \geq \prod_{j=1}^{c(x)+1} 2^{-\hat{L}_j} \tag{3}$$

This inequality arises because:

- The number of valid options at each step is reduced, as previously used integers are no longer available.
- The probability of generating a valid integer becomes higher than $2^{-\hat{L}_j}$ for each phrase.
- Consequently, the overall probability of generating the sequence x is higher than what would be calculated by the product of $2^{-\hat{L}_j}$.

The inequality still provides a lower bound on the probability, which is sufficient for proving the exponential order of $2^{-c(x)\log c(x)}$. Thus, despite the modification, the core conclusion about the algorithm's alignment with the LZ universal probability distribution remains intact.

$$\begin{aligned}
\tilde{P}(x) &\geq \prod_{j=1}^{c(x)+1} 2^{-\hat{L}_j} \\
&= \exp_2\left\{-\sum_{j=1}^{c(x)+1} \hat{L}_j\right\} \\
&\geq \exp_2\left\{-\sum_{j=1}^{c(x)+1} L_j\right\} \\
&\geq \exp_2\left\{-\sum_{j=1}^{c(x)+1} \log(2j\alpha)\right\} \\
&\geq \exp_2\left\{-[c(x)+1]\log[2c(x)\alpha]\right\} \\
&= \exp_2\left\{-[c(x)+1]\log c(x) - [c(x)+1]\log(2\alpha)\right\}.
\end{aligned} \tag{4}$$

B. Algorithm A_2: Learning from a Prefix

This algorithm extends the basic concept by utilizing the first l elements of x, to learn the relationship between x and y.

This algorithm performs parsing and encoding of the first l elements of x, given y, as described in algorithm 3 and then performs the guessing for the rest of the sequence in the same manner as described in algorithm A1.

Algorithm A2 Universal Guessing with Partial Information

```
encode the first l symbols of x and y like described in algorithm 3
Initialize D_x, D_I, D_L using the encoded results
end ← l
while not end of y do
    beg ← end
    L ← Generate_L(min_L, max_L, D_x, D_L)
    end ← beg + L
    phrase ← y[beg : end]
    if phrase in D_x then
        output D_x[phrase]
    else
        I ← Generate_I(D_I, D_x)
        I_a ← I mod α
        π ← (I − I_a)/α
        if π = 0 then
            add I_a to D_x
        else
            Look up phrase p corresponding to π in D_x
            add p concatenated to I_a to D_x
            output p concatenated to I_a
        end if
    end if
    add L and I to D_I, D_L
    min_L, max_L ← UpdateBounds(L, D_L)
end while
```

C. Algorithm A_3: Generating Based on Known Parsing

Our third variant adopts a new approach by only using joint phrases observed in the known part of x when generating guesses for the unknown part. It is similar to algorithm A2, only it doesn't update the dictionaries in the guessing part, and keeps on referring only to the phrases created in the encoding section. for each new guessing of a phrase, we look for the longest prefix of y in the known parsing, and guess the matching phrase of x. Notice that this algorithm no longer outputs a sequence matching LZ decoding, since now the same phrase can appear multiple times.

Algorithm A3 Universal Guessing Based on Known Parsing

encode the first l symbols of x and y like described in algorithm 3
Initialize D_x, D_I, D_L using the encoded results
Set max_L to be the longest observed phrase
end $\leftarrow l$
while not end of y **do**
    beg $\leftarrow$ end
    $L \leftarrow$ max_L
    while $L > 1$ **do**
        end $\leftarrow$ beg $+ L$
        phrase $\leftarrow y$[beg : end]
        if phrase$[:-1]$ in D_x **then**
            break
        end if
        $L \leftarrow L - 1$
    end while
    $I \leftarrow$ Generate_I(D_I, D_x)
    $I_a \leftarrow I \bmod \alpha$
    $\pi \leftarrow \frac{I - I_a}{\alpha}$
    if $\pi = 0$ **then**
        add I_a to D_x
    else
        Look up phrase p corresponding to π in D_x
        output p concatenated to I_a
    end if
end while
output guessed phrases of x by order

1) Key Challenges and Solutions:

- Context-Based Length Selection: In A_3, we select the largest L from the observed context and decrease it until a valid phrase is found. This approach leverages the learned patterns more effectively than random selection.

D. Proof of equivalence

Let X be a random variable such that $X = (X_0^l, X_l^n)$, where X^l denotes the first l elements of X and X_l^n denotes the remaining elements. Let Y be another random variable. Notice the following inequality:

$$H(X|Y) \geq H(X|X_0^l, Y)$$

This inequality holds since conditioning reduces entropy.

IV. Results

In this section, we evaluate the performance of the universal guessing algorithms under various conditions.
Algorithm A_1 relies solely on side information without any knowledge about x. it consistently achieved around 50% accuracy, equivalent to random guessing. That is due to the fact that the algorithm isn't provided with any information about x, and is unable learn the correlation between x and y. Our analysis focuses on algorithms A_2 and A_3, as they utilize partial information about the sequence x.

Our experimental setup is as follows:

1) We generate binary sequences x and z according to specified distributions.
2) We create the side information sequence y by applying the XOR operation: y = x $\oplus$ z.
3) We provide the algorithms with partial information about x (a prefix of known length l) and the entire sequence y.
4) The algorithms attempt to learn the relationship between x and y, which in our case is determined by the noise sequence z.
5) Using this learned relationship, the algorithms attempts to guess the future bits of x.

By varying the distributions of x and z, as well as the length of the known prefix, we can assess the algorithms' performance and adaptability across different conditions.

To evaluate the performance of our algorithms, we employ two main metrics:

1) **Segmented Accuracy**: We divide the sequence into segments of fixed length and calculate the percentage of correctly guessed bits within each segment. This metric allows us to observe how the accuracy of the algorithms changes over time, that is, throughout the sequence.
2) **Overall Accuracy**: We calculate the total percentage of correctly guessed bits across the entire sequence, excluding the known prefix. This metric provides a single value representing the algorithm's overall performance.

A. Majority Voting

Before presenting our main results, we introduce a technique to improve the accuracy of our guessing algorithms: majority voting.

Majority voting is a method where we run the guessing algorithm multiple times and take the most frequent outcome for each bit as the final guess. This approach can help reduce the impact of random errors and improve overall accuracy.

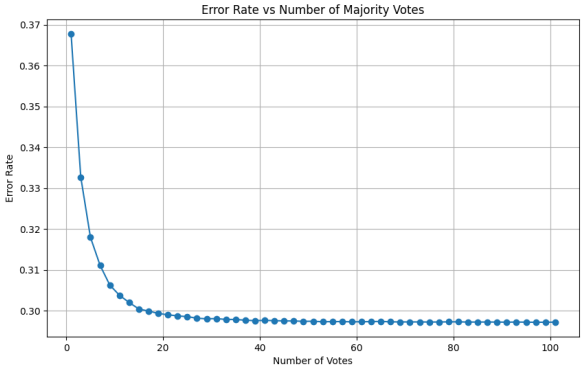

Fig. 7. Error Rate vs Number of Votes, for i.i.d process with $p_x = 0.6$ and i.i.d noise with $p_z = 0.3$.

Figure 7 shows how the error rate decreases as we increase the number of votes in the majority voting process. on an i.i.d process with $p_x = 0.6$ and i.i.d noise with $p_z = 0.3$. Key observations:

- The error rate drops rapidly, showing significant improvement with just a few iterations.
- After about 20-30 votes, the improvement rate slows down, suggesting diminishing returns for additional votes.
- The error rate stabilizes around 0.297, indicating the limit of improvement possible through majority voting for this particular scenario.

Majority voting can fasten the convergence to the optimal prediction. it also improves the decay of accuracy in A2. from now on, all of the results we present will use majority voting.

B. Performance Comparison of A2 and A3 For Binary Sequences

1) i.i.d Sequences with Low Noise: This is our standard case of estimation, we have a relatively low noise, and we try to guess x from learning the connections between x and y.

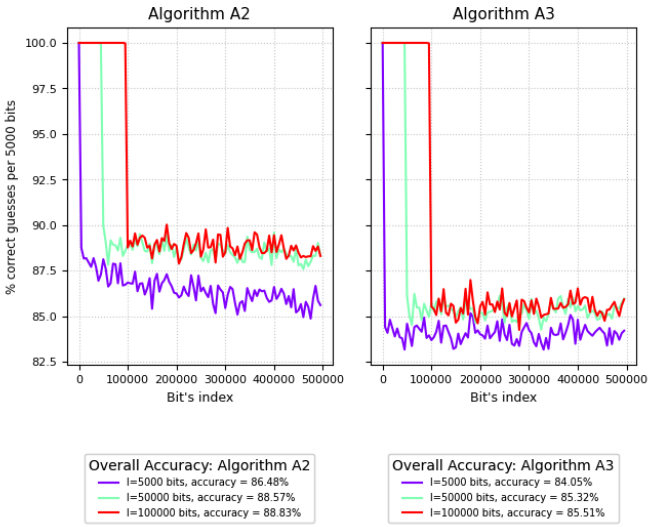

Fig. 8. Guessing algorithms Comparison. x is an i.i.d sequence with probability $x_i \sim \text{Ber}(0.9)$, given side information $y = x \oplus z$, where z is an i.i.d noise sequence with probability $z_i \sim \text{Ber}(0.1)$. The length of the known prefix of x is denoted by l

2) i.i.d Sequences with High Noise: In this case, when we look at the values of y, we know nothing about x, since in this case x and y are independent. therefore, if we manage to learn the distribution of x we do so from the prefix of x alone. Our results show that even with no correlation between x and y, if x's distribution is extreme enough, we can indeed learn it.

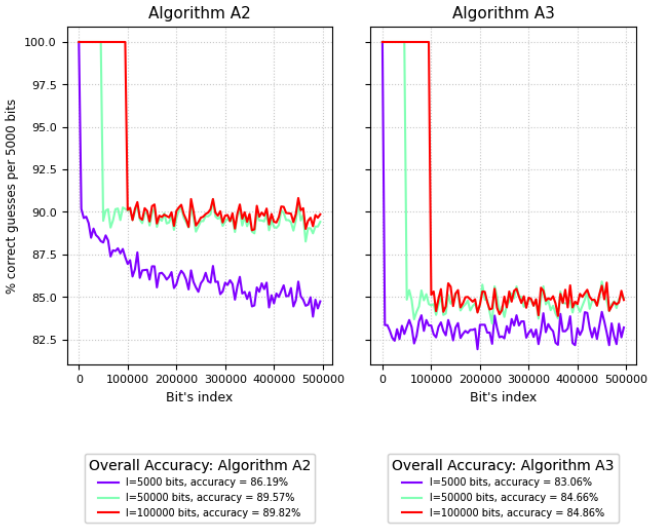

Fig. 9. Guessing algorithms Comparison. x is an i.i.d sequence with probability $x_i \sim \text{Ber}(0.9)$, given side information $y = x \oplus z$, where z is an i.i.d noise sequence with distribution $z_i \sim \text{Ber}(0.5)$. The length of the known prefix of x is denoted by l

3) Markov 2nd Order Sequence and i.i.d Noise: Now we use a different distribution for generating x. particularly,

x is a second order Markov chain with transition matrix $\begin{pmatrix} 0.3 & 0.7 \\ 0.7 & 0.3 \\ 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$, and a stationary probability $x_{-1}, x_0 \sim \text{Ber}(\frac{1}{2})$. z is an i.i.d noise with distribution $z_i \sim \text{Ber}(0.1)$ In this case, we get the following results:

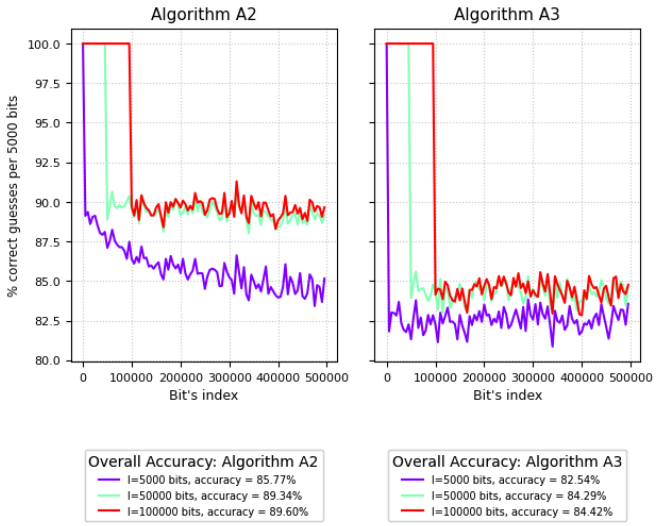

Fig. 10. Guessing algorithms Comparison. x is a second order Markov chain with transition matrix $\begin{pmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \\ 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$, and a stationary probability $x_{-1}, x_0 \sim \text{Ber}(\frac{1}{2})$, given side information $y = x \oplus z$, where z is an i.i.d noise sequence with probability $z \sim \text{Ber}(0.1)$. The length of the known prefix of x is denoted by l

4) i.i.d x With a First Order Markov Chain Noise: Finally, we want to test out algorithms for more complicated noise sequences. We tested The algorithms for an i.i.d x, with probability $x_i \sim \text{Ber}(0.9)$, and for a first order Markov chain z, with transition matrix $\begin{pmatrix} 0.3 & 0.7 \\ 0.7 & 0.3 \end{pmatrix}$, and a stationary probability $z_0 \sim \text{Ber}(\frac{1}{2})$

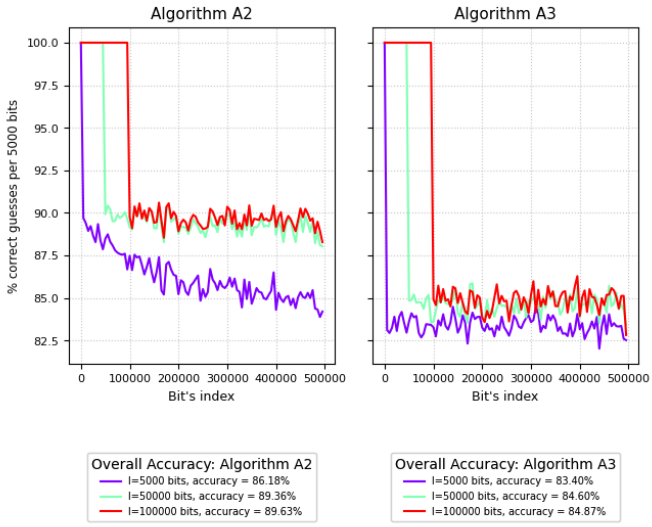

Fig. 11. Guessing algorithms Comparison. x is an i.i.d sequence with probability $x_i \sim \text{Ber}(0.9)$, given side information $y = x \oplus z$, where z is a first order Markov chain with transition matrix $\begin{pmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{pmatrix}$, and a stationary probability $z_0 \sim \text{Ber}(\frac{1}{2})$. The length of the known prefix of x is denoted by l

5) Analysis:

- Both A2 and A3 show improved performance as the known prefix length l increases.
- A2 exhibits a wider range of performance across different values of l compared to A3.
- Algorithm A2 shows accuracy decay. This is due to error propagation- every time we guess a new phrase, we add it to the dictionary, even if the guess is incorrect, so later on in the guessing process, we can choose this phrase as a prefix and propagate an error.
- A3 shows a more consistent performance across different values of l, with less variance in accuracy.
- For larger values of l, A2 slightly outperforms A3.
- The proposed guessing method operates without requiring precise knowledge of the event probabilities. Instead, it achieves high accuracy by selecting the maximum value from the set of probabilities and their complements. Specifically, the method guarantees an accuracy of $\max(\max(P_X, 1-P_X), \max(P_Z, 1-P_Z))$. This approach enables the method to perform comparably to algorithms that are aware of the exact probabilities, particularly in cases where there are dominant probabilities.

C. Performance Comparison of A2 and A3 For Image Reconstruction

To demonstrate the practical applications of our guessing algorithms, we extended their use to image denoising. We applied both A2 and A3 algorithms to reconstruct noisy binary

images under two different noise scenarios: image-based noise and i.i.d noise.

Our image reconstruction process followed these steps:

- We converted a grayscale image to a binary sequence, where each pixel is represented by a single bit (0 for black, 1 for white).
- We added noise to the sequence using two methods:
  - Image-based noise: We used a separate noise image and applied it to the original image using a bitwise XOR function.
  - i.i.d noise: We flipped bits with probability $p_z = 0.3$.
- We provided the algorithms with the noisy image as side information y and the first 20% of the original image as the known prefix.
- The algorithms attempted to reconstruct the remaining bits of the original image.

Fig. 12. Original Image- a binary image of size 512x512 pixels

1) i.i.d Noise Reconstruction: First, we applied algorithms $A2$ and $A3$ to on the image, using i.i.d noise ($p_z = 0.3$).

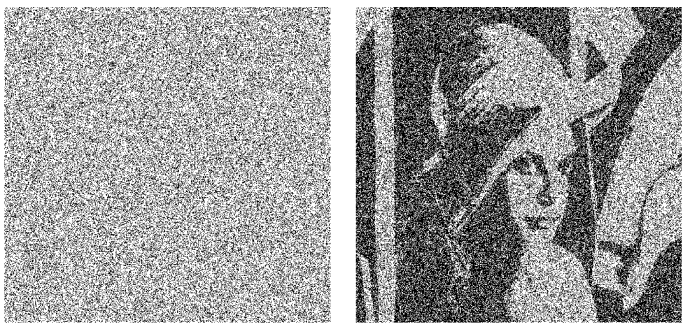

Fig. 13. i.i.d noise with $p_z = 0.3$ and resulting noisy image

The results were as follows:

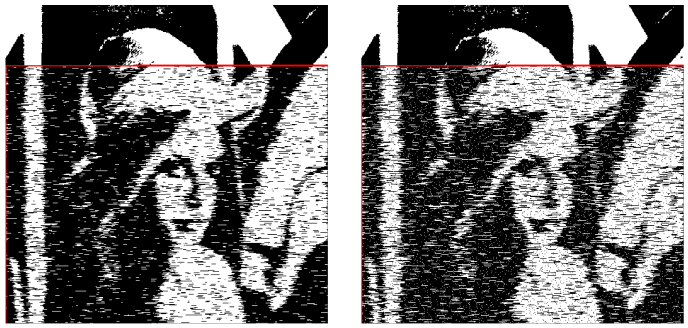

Fig. 14. Image reconstruction results with i.i.d noise: (Left) A2 reconstruction, (Right) A3 reconstruction, given a noisy image and the first 20% of the image (marked in red).

- Algorithm A2 (Figure 14, left) achieved an overall accuracy of 84.47%.
- Algorithm A3 (Figure 14, right) achieved an overall accuracy of 79.79%.
- These results are relatively good. The noise probability is $p_z = 0.3$, so given the distribution of the noise, a standard estimation would achieve an accuracy of about 70%. Yet, even though our algorithms have no prior knowledge of the distribution, they still achieve a higher accuracy percentage.

Fig. 15. Image reconstruction results with i.i.d noise: (Left) A2 reconstruction, (Right) A3 reconstruction, given a noisy image and the first 20% of the image, using Hilbert space filling curve as the vectorization scheme

- Algorithm A2 (Figure 14, left) achieved an overall accuracy of 85.3%.
- Algorithm A3 (Figure 14, right) achieved an overall accuracy of 80.4%.
- using the Hilbert curve did not significant help with accuracy. but the reconstructed image gained a different noise characteristic which could help when employing further denoising techniques.

2) Image-based Noise Reconstruction: We tested our algorithm on a noise image with an unknown distribution. The noise and the noisy image are presented in figure 16

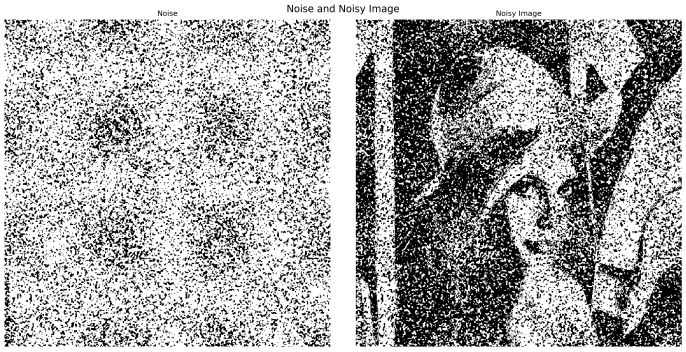

Fig. 16. Image-based noise and resulting noisy image

- The noise pattern (Figure 16, left) shows a complex, non-stationary distribution.
- The noisy image (Figure 16, right) shows significant degradation, making many features of the original image difficult to discern.

The reconstruction results show:

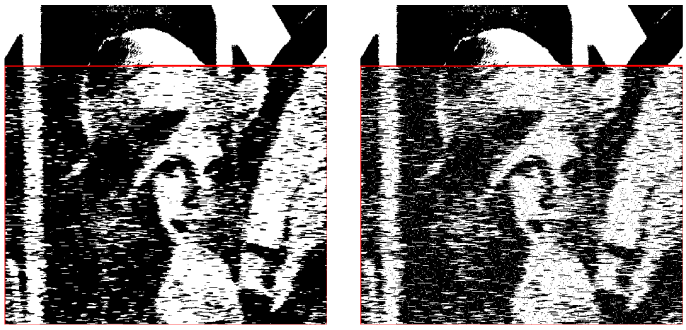

Fig. 17. Image reconstruction results with image-based noise: (Left) A2 reconstruction, (Right) A3 reconstruction

Figure 17 presents the results of our image reconstruction experiment using image-based noise.

- Algorithm $A2$ (Figure 17, left) achieved an overall accuracy of 86.42%.
- Algorithm $A3$ (Figure 17, right) achieved an accuracy of 80.42%.
- Again, algorithm $A2$ performed better. The difference is most apparent in the areas most affected by the noise.

D. Conclusions

When comparing the results of both of the algorithms, A3 shows slight advantages at shorter prefix lengths and A2 performs marginally better at longer prefix lengths. This suggests that the two algorithms have different strengths depending on the amount of available information. A2 may be preferable in applications where maximum accuracy is crucial and longer training sequences are available, while A3 might be more suitable in scenarios requiring consistent performance across varying conditions. For image denoising applications, the choice between A2 and A3 may depend on the expected noise levels in the input images.

Overall, these results helps us determine the importance of prefix length in estimation for the different algorithms, and see that the sequence estimations converge to what's expected give the information. At the task of image reconstruction, we managed to get a better accuracy than just elementary denoising on variety of noise conditions we tested, based on a relatively small amount of information.

Future work could explore the theoretical foundations of these performance differences and investigate potential hybrid approaches that combine the strengths of both algorithms, and showing results for a different alphabet sequence.

REFERENCES

[1] N. Merhav and A. Cohen, "Universal randomized guessing with application to asynchronous decentralized brute–force attacks," *IEEE Transactions on Information Theory*, vol. 66, no. 4, pp. 2651–2671, 2020.

[2] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[3] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley-Interscience, 2nd ed., 2006.

[4] T. Uyematsu and S. Kuzuoka, "Conditional lempel-ziv complexity and its application to source coding theorem with side information," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E86-A, no. 10, pp. 2615–2617, 2003. Special Section on Information Theory and Its Applications.