LEGO: Learnable Groupings for Lossless Compression

Daniel Severo
University of Toronto
Vector Institute for Artificial Intelligence
d.severo@mail.utoronto.ca

Abstract

Most off-the-shelf data compression software combine multiple transformations and lossless coders to achieve greater compression rates. There has been a surge in the use of machine learning to estimate the symbol probabilities for entropy coders. However, these methods require amortizing over large datasets and training huge latent variable models. On the other hand, general purpose compressors (GPCs) such as run-length encoding do not require an estimate of the probability mass function and can work with arbitrary sized sequences. GPCs achieve even greater performance when proceeded by transformations such as the Burrows-Wheeler (BWT). In this work we exploit recent developments in gradient estimation of discrete combinatorial objects to learn data-dependent transformations that incur greater savings than BWT when used with run-length encoding. Our methods are comparably fast, as they forgo the use of a model and instead directly optimize the cost matrix of a linear assignment problem through the Sinkhorn-Operator.

1. Introduction

Data compression algorithms are pervasive throughout literature and come from two main design philosophies (Steinruecken, 2015). *Probabilistic compressors* (PCs) such as Huffman coding, arithmetic coding (AC) (Cover, 1999), and asymmetric numeral systems (ANS) (Duda, 2013), require an estimate of the source probability mass function (PMF). However, *general-purpose compressors* (GPCs) like Unary, Run-Length (RLE) (Robinson & Cherry, 1967), Elias Gamma (Elias, 1975), and the Lempel-Ziv (Welch, 1984) coding algorithms operate directly on the sequences, which do not require the PMF.

More recently, there have been developments in PCs that leverage the success of latent variable models to achieve better compression rates by accurately estimating the data generating distribution together with bits-back coding (Frey & Hinton, 1996; Kingma, 2019; Ruan et al., 2021; Townsend et al., 2019a;b). Although promising, these methods work in a setting which is not practical in many application. Most notably, they require training large models on datasets which provide enough examples for amortization of the message length. Second, most are dataset-specific, and require retraining in order to be used with new sources.

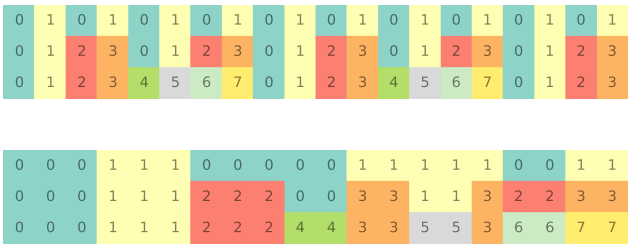

Figure 1. A dataset of 3 input sequences of length 20 (top) is transformed to a LEGO block of the same size (bottom), which has significantly more runs. The permutation matrix is 20x20 and is shared by all input sequences. Equivalently, since the same permutation matrix is applied to all 3 sequences, this can be viewed as a permutation of the columns of the input. The LEGO is flattened in row major order before being compressed with run-length encoding.

GPCs, on the other hand, do not assume knowledge of the data generating distribution. Instead, they operate mechanistically on the sequences emitted by the source. For GPCs, the order of symbols in a sequence can significantly change the code-length. For example, RLE encodes a sequence AAAAABAA as its frequency count 5A1B2A, while the permutation $T(\text{AAAAABAA}) = \text{AAAAAAAB}$ has a shorter encoding 7A1B. A good choice of T can significantly reduce the average code-length, provided it is invertible and available both during compression and decompression. Notable examples are the Move-to-front (MTF) (Ryabko, 1980) and Burrows-Wheeler (BTW) (Burrows & Wheeler, 1994) transformations, which are heuristics that transform the input sequence without taking into account the characteristics of the source.

In practice, successful GPCs chain multiple transformations and entropy coders. For example, bzip2 (Seward, 1996) first applies RLE, followed by BWT, MTF and then RLE again, before proceeding to Unary and Huffman coding. This brings us to the central question of our proposal: *can invertible transformations for lossless compression be learned from data?*.

In this work we investigate the question above and propose two algorithms called LEGO and LEGO+ that reduce the final message length when used together with GPCs. They work by using recent ideas for gradient estimation of discrete variables by deriving soft-relaxations (Jang et al., 2016; Maddison et al., 2016) of a linear assignment problem through the Sinkhorn-Operator (Adams & Zemel, 2011; Mena et al., 2018). Our methods are model-free, in the sense that they do not use a neural network. Instead, they directly optimize the cost matrix of the linear assignment problem(Kuhn, 1955).

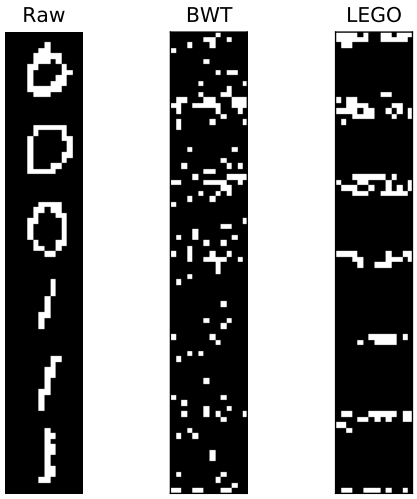

Figure 2. BMNIST(0,1). The images are down-sampled with a scale factor of $1/2$. BWT (middle) and LEGO (right) where applied at the image level, and the dataset was flattened in row major order to be compressed with RLE.

2. Background

2.1. Lossless Compression

Lossless compression aims to find a minimal and perfectly invertible representation of the data. Formally, given a discrete source of alphabet size m that outputs sequences $x^n = x_1 x_2 \dots x_n \in \{1, \dots, m\}^n$ sampled from a distribution P, the objective is to find a *code* C from $\{1, \dots, m\}^n$ to the set of all possible binary strings of finite length $\{0, 1\}^\star$. Using $\ell(x^n)$ to denote the size of the binary string $C(x^n)$, we can write the *average code-length*

$$\mathbb{E}[\ell(x^n)] = \sum_{x^n} P(x^n)\ell(x^n), \tag{1}$$

which represents the average number of bits needed to store sequences x^n using code C. Shannon's Source Coding theorem states that no code can achieve an average code-length smaller than the *entropy* of the source $H_n(P) = -\mathbb{E}[\log_2 P(x^n)]$ (Cover, 1999). This provides a fundamental lower bound on the achievable performance of lossless compression. A code that achieves the entropy is called a *perfect entropy coder*.

2.2. Run-length Encoding

Run-length Encoding (RLE) (Robinson & Cherry, 1967) is a lossless data compression algorithm that stores sequences as a count of consecutive runs. For example, the sequence $AAABBAAAC$ is represented as the *run-message* $3A2B3A1C$. The ordered list of counts, $[3, 2, 3, 1]$, are referred to as the *run-lengths*; while the list of symbols, $[A, B, A, C]$, are the *run-symbols*. In this example, we say that the sequence consists of 4 runs.

For storage, the run-message must be encoded using some binary representation, which is a non-trivial problem. The precision of the run-lengths must be carefully tuned to allow large runs without overshooting for precision. For example, suppose we chose to represent the run-lengths as unsigned-integers of 8 bit precision. Then, runs larger than $2^8 = 256$ must be broken down into 2 or more separate runs. A similar problem occurs for the run-symbols. For small sequences, the aforementioned problem is less salient. In this work we deal only with the number of runs, i.e. the size of the run-lengths list, which is a good proxy for the final length of the compressed message.

RLE works best on sequences that have large patches of constant symbols, as this produces larger and less runs. The quintessential examples are lossless compression for scanned documents and antique analog television signals.

2.3. Burrows–Wheeler Transform

The Burrows-Wheeler transform (BWT) (Burrows & Wheeler, 1994) is a deterministic and reversible transformation that is used in compression pipelines to achieve better compression rates. It is best explained through an example. Consider the sequence 123123. First, create the matrix of all possible shifts (starting with the sequence itself) and sort

the rows lexicographically,

$$\begin{pmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 3 & 1 & 2 & 3 & 1 & 2 \\ 2 & 3 & 1 & 2 & 3 & 1 \\ 1 & 2 & 3 & 1 & 2 & 3 \\ 3 & 1 & 2 & 3 & 1 & 2 \\ 2 & 3 & 1 & 2 & 3 & 1 \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 2 & 3 & 1 & 2 & \mathbf{3} \\ 1 & 2 & 3 & 1 & 2 & \mathbf{3} \\ 2 & 3 & 1 & 2 & 3 & \mathbf{1} \\ 2 & 3 & 1 & 2 & 3 & \mathbf{1} \\ 3 & 1 & 2 & 3 & 1 & \mathbf{2} \\ 3 & 1 & 2 & 3 & 1 & \mathbf{2} \end{pmatrix}.$$

Return the last column (in bold) together with the index $I = 1$ that indicates the row containing the original sequence.

Decoding is done by noting that all columns are permutations of the original sequence, and the first column can be recovered by sorting the received message (i.e. column in bold). Together, the first and last columns form all possible 2-gram tokens of the original sequence, and this can be used to reconstruct the entire matrix. The index $I = 1$ is then used to recover the input.

The output sequence has more consecutive repeating symbols (i.e. runs), which facilitates compression. To understand how this is achieved, note that the lexicographical sort is equivalent to sorting all 2-gram tokens, with ties being broken by n-grams of larger order. Outputing the last column exploits the Markovian structure of the input sequence.

2.4. Sinkhorn-Operator

In our context, the Sinkhorn-Operator S (Adams & Zemel, 2011) is a function that iteratively normalizes rows and columns of a matrix X; which can be used to learn permutations through gradient optimization.

To understand this, consider the linear assignment problem with cost matrix X and solution $M(X)$, both $n \times n$. Using notation from (Mena et al., 2018),

$$M(X) = \arg\min_{P \in \mathcal{P}} \langle P, X \rangle, \tag{2}$$

where $\langle P, X \rangle = \sum_{i,j} P_{ij} X_{ij}$ is the matrix norm, and $\mathcal{P}$ the set of all permutation matrices of size $n \times n$. In the traditional setting, the cost matrix X would be given and the solution $M(X)$ could be found using a solver such as the Hungarian Algorithm (Kuhn, 1955). However, we may also view $M(X)$ as parameterized by X, in the sense that we can learn a permutation $M(X)$ by modifying X to maximize some objective.

Unfortunately, the gradient signal of $M(X)$ with respect to X is discontinuous due to the $\arg\min$ operator, which precludes the direct use of backpropagation and other gradient methods. Luckily, it was shown in (Mena et al., 2018) that $M(X) \to S(X/\tau)$ as $\tau \to 0^+$, where τ is known as a temperature parameter. The proof follows from showing that $S(X/\tau)$ is the solution of an entropy-regularized problem in the set of doubly-stochastic matrices $\mathcal{B}$,

$$S(X/\tau) = \arg\min_{P \in \mathcal{B}} \langle P, X \rangle + \tau h(P), \tag{3}$$

where $h(P) = -\sum_{i,j} P_{ij} \log P_{ij}$ is the entropy of P. As $\tau \to 0^+$, the solution is pushed to a vertex of $\mathcal{B}$, which is a permutation matrix.

In theory, this setup is still not usable for gradient optimization, as the Sinkhorn-Operator requires infinitely many iterations. However, it has been shown empirically that a finite amount provides meaningful gradients for backpropagation.

3. LEGO for Lossless Compression

3.1. Problem Setting

Our setting is that of learning transformations for sequences $x^n \in \mathcal{X}^n$ emitted from an unknown source. No assumption is made regarding the data generating distribution. We assume sequences have a fixed shape and share a common structure of some nature. For example, consider the flattened representation of images from BMNIST, which has shape $n = 784$. The common structure is implied through the fact that all sequences are representations of digits, which becomes stronger if we filter based on a common label (e.g. use only 0's).

We assume access to some lossless compression algorithm C capable of encoding and decoding sequences of arbitrary length k, such as RLE. We accumulate a set $\{x_i^n\}_{i=1}^m \in \mathcal{X}^{m \times n}$ of sequences $x_i^n \in \mathcal{X}^n$ emitted from the source, called a *block* (see Figures 1 and 3 for examples of blocks). The objective is to learn a transformation over the blocks, $\mathcal{T} : \mathcal{X}^{m \times n} \to \mathcal{X}^k$, that results in a smaller code-length once compressed with C. Formally,

$$\ell_C(\mathcal{T}(\{x_i^n\}_{i=1}^m)) < \ell_C(\{x_i^n\}_{i=1}^m), \tag{4}$$

where $\ell_C(z)$ returns the length of code-word $C(z)$. In this work, we consider transformations that are permutations over the individual input sequences ($k = n$) or the entire block ($k = mn$). The permutation $\mathcal{T}$ must be encoded together with the input, so that we may un-permute to recover the original sequence during decoding.

As an example, consider the top block of Figure 1. There, $m = 3$, $n = 20$, and $\mathcal{X} = \{0, 1, \ldots, 7\}$. The bottom block is the output of LEGO+, discussed in the following section.

In this work, we fix C to be a vanilla RLE algorithm. As discussed in section 2.2, we use the number of runs as a proxy for the compressed code-length ℓ_{RLE}.

3.2. LEGO

LEGO is an algorithm that learns groupings of block columns, in a way that reduces the number of runs. Following subsection 2.4, we learn a permutation by solving a linear assignment problem.

Learning permutations over entire blocks incurs an overhead of $\mathcal{O}((mn)!)$, due to the necessity of adding $\mathcal{T}$ to the message. This can be reduced by learning permutations that factorize over sequences,

$$\mathcal{T}(\{x_i^n\}_{i=1}^m) = \{\mathcal{T}(x_i^n)\}_{i=1}^m, \tag{5}$$

resulting in an overhead of $\mathcal{O}(n!)$. In the extreme case (which we found performs best) only one permutation matrix per block is learned. This allows us to directly apply back-propagation on the cost matrix X, forgoing the use of a model (e.g. neural network); and has the added benefit of converging quickly. Sharing the permutation matrix across sequences in a block is equivalent to permuting the column of the block, which is the etymological origin of LEGO.

Overall, this results in an equalizer strategy that minimizes

$$\sum_{i=1}^{m} \ell_{\text{RLE}}(Px_i^n), \tag{6}$$

where P is some permutation matrix, and x_i^n a column vector representing the input sequences in a block. This problem is converted to a supervised setting by designing target sequences $y_i^n = \text{sort}(x_i^n)$ and minimizing

$$\sum_{i=1}^{m} \|S(X/\tau)x_i^n - y_i^n\|^2, \tag{7}$$

with respect to X through backpropagation; where $S(X/\tau)$ is the soft-permutation matrix (Sinkhorn-Operator) with temperature τ. Note that (7) is used for training, but to compute the permutation matrix we apply the Hungarian Algorithm (Kuhn, 1955) to X/τ.

3.3. LEGO+

LEGO+ modifies the loss function of LEGO through data-augmentation to account for the non-uniqueness of optimal targets y^n.

The problem of constructing y^n is non-trivial, as it is compression-algorithm-dependent. For a block size of $m = 1$, the target y^n should be that which minimizes the code-length for the given compressor C, which may not be unique. For RLE, we conjecture that the set of optimal y^n can be constructed by permuting the runs of x^n after sorting. For example, if $x^n = 12123$ then any $y^n \in R(x^n) = \{11223, 11322, 22113, 22311, 31122, 32211\}$ will have 3 runs, which is the minimum possible. For $m \geq 2$, there is no guarantee that this is a valid strategy, as the permutation matrix is shared across sequences in the block.

Despite this, we experimented by changing the loss function to,

$$\sum_{i=1}^{m} \sum_{y_i^n \in R(x^n)} \|S(X/\tau)x_i^n - y_i^n\|^2, \tag{8}$$

which gives rise to ELBO+.

4. Experiments

In this section, we present a selection of 12 experiments that best represent our empirical findings. We implemented vectorized versions of both RLE and BWT which are publically available[1]. Experiments follow the same structure. First, a transformation (BWT, LEGO or LEGO+) is applied to the sequence, which is then flattened in row major order before being compressed with RLE to analyze the total number of runs.

In all cases, LEGO and LEGO+ were trained for 20 and 30 iterations[2], respectively. The learning rate was fixed to 1, while the temperature parameter was 0.5. Convergence of the Sinkhorn-Operator was not sensitive to the number of iterations (i.e. normalization of columns and rows), which was fixed to 3. BWT requires no training and has no parameters. However, the complexity is linear with respect to sequence size.

Results are shown in Table 1. Four sources are used: Lattice, Uniform, BMNIST(0) and BMNIST(0, 1), which are explained in their respective subsections below.

On all datasets, we performed an experiment by using a block of size $m = 1$ to see if the algorithms are able to learn the ideal target sequence y^n. In general, we found that LEGO and LEGO+ succeeded on all datasets, while BWT did so only on Lattice. See Figure 4 for an example.

Due to the pervasive nature of machine learning methodology, we'd like to remind the reader that this is *not* a traditional machine learning setting. First, the block sizes of our experiments are small. For example, the largest input blocks were BMNIST(0) and BMNIST(0,1), both with a shape of $(6, 14, 14)$. Meanwhile, the original BMNIST test set is $(10000, 28, 28)$. Similarly, there is no notation of train and test sets, as LEGO, LEGO+, and BWT are applied *on a block of sequences to compress that same block*.

[1] https://gist.github.com/dsevero/693677754798e21f539e4e11a3103576

[2] If we consider the block as the dataset, then this becomes the number of epochs.

4.1. Lattice

In our context, a Lattice block of periods $\{p_i\}_{i=1}^m$ and length n is made up of sequence

$$x^n = \{i \mod p_i\}_{i=1}^n, \tag{9}$$

where $\mod$ is modulo division. See the top block of Figure 1 for an example with periods $\{2, 4, 8\}$ and length $n = 20$. The block size m is automatically determined by the number of periods.

BWT performs very well on this source as can be seen in Table 1, where it outperforms all other algorithms by a significant margin. To understand why, please see section 2.3 and note that the example sequence is an offset Lattice with $n = 6$ and a single period equal to 3. Both LEGO and LEGO+ are able to reduce the number of runs, but are still 3 times worse than BWT.

4.2. Uniform

This source is generated by uniformly sampling integers in a given range $[a, b]$. See Figure 3 for an example with $a = 0, b = 5$.

None of the algorithms are able to perform well, and the reduction in number of runs is insignificant. In some cases for BWT (not shown in Table 1), the number of runs increased. This makes sense, as the source is designed to ensure no structure between sequences due to the uniform sampling. However, LEGO and LEGO+ never increased the number of runs.

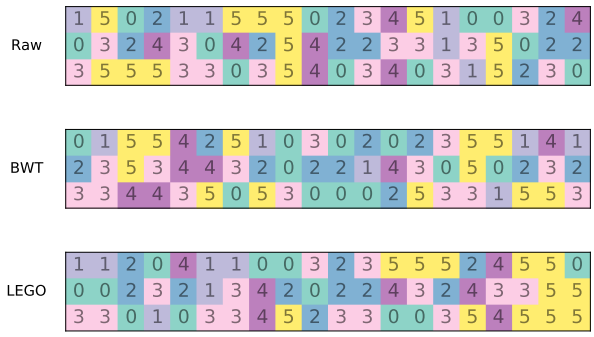

Figure 3. Uniform source. Raw (top) is constructed by uniformly sampling integers between 0 and 5. Neither BWT (middle) nor LEGO (bottom) learn relevant transformations that can significantly decrease the number of runs.

4.3. BMNIST(0) and BMNIST(0,1)

This source emits sequences that are flat representations of BMNIST. In both datasets, the images are first downsampled by a scale factor of $1/2$. Sequences always have length $n = 14 \times 14 = 196$ and block size $m = 6$. The number inside parenthesis indicates the digit labels used. BMNIST(0) emits only images corresponding to digit 0, while BMNIST(0,1) mixes 0 and 1 and can be seen in Figure 2.

Table 1. LEGO+ outperforms BWT on BMNIST variants and Uniform sources with respect to number of runs. BMNIST(0,1) is shown in Figure 2, while BMNIST(0) is similar but uses only digits of label 0. Lattice source is shown in Figure 1, while Uniform is similar to Figure 3 but with a larger block size equal to 10. Run savings are that of LEGO+ with respect to Raw.

	BMNIST(0)	BMNIST(0,1)	Uniform	Lattice
Raw	137	157	170	60
BWT	171	197	171	**11**
LEGO	89	116	160	36
LEGO+	**85**	**114**	**155**	35
Savings	38%	27%	9%	-

Both LEGO+ and LEGO perform well in this setting, while BWT does not and mostly increases the number of runs due to the lack of Markov structure between pixel sequences. Results on BMNIST(0) are better than BMNIST(0,1), which is expected due to the stronger structure between images (only 0 labels vs 0's and 1's).

5. Limitations

A major limitation in our current algorithms is the lack of a proper method to account for the non-uniqueness of the optimal target sequences. While LEGO+ partially addresses this, the increase in savings is not significant. Another issue is scaling. All methods, BWT, LEGO and LEGO+ failed to learn meaningful permutations for BMNIST-like sources without down-sampling (i.e. sequence length $n = 784$). A possible extension is to use BWT to create the optimal target sequences. To address scaling issues, incremental learning could be explored where we train sequences of BMNIST-like sources with a schedule on the scale factor.

6. Related Work

To the best of our knowledge, learning permutations for already existing GPCs has not been previously investigated. However, (Parent & Nowe, 2002) propose a genetic algorithm that evolves to complex transformations from basic pre-defined primitive operations. The fitness measure is the negative empirical entropy of the sequence itself. GP-ZIP (Kattan & Poli, 2008) is another genetic program that evolves to optimally toggle between different entropy coders and transformations during compression. A handful of hand-crafted transformations have been proposed over time. A survey is provided at (Radescu, 2009).

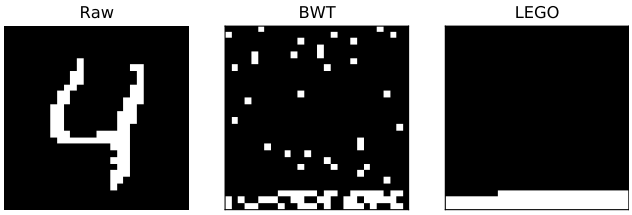

Figure 4. A single BMNIST image (left) is perfectly sorted by LEGO (right), but not by BWT (middle). No down-sampling was applied in this case.

7. Conclusion

In this work, we show that it is possible to achieve better compression rates with off-the-shelf general purpose compressors by learning a transformation on the input sequences. We developed two algorithms that learn permutations by casting them as the solution to a linear assignment problem, following the work of (Mena et al., 2018). Our methods are called LEGO and LEGO+, and make no assumptions regarding the data generating distribution. Instead, we search for structure that can be exploited between sequences.

LEGO+ exploits the non-uniqueness of optimal target sequences through data-augmentation to achieve better compression rates. We forgo the use of a model, and directly optimize the cost matrix via gradient descent and the Sinkhorn-Operator (Adams & Zemel, 2011).

Compression performance is evaluated by measuring the number of runs resulting from applying the transformation before doing run-length encoding. We benchmark our algorithms against the Burrows-Wheeler Transform (BWT). Overall, we found that LEGO+ outperforms all other methods on sources constructed from benchmark datasets such as BMNIST. When no structure between sequences is present (such as in the Uniform source), BWT, LEGO and LEGO+ fail to improve the compression rate. LEGO and LEGO+ are able to learn an ideal permutation when the block size is $m = 1$, but BWT does not achieve this consistently.

8. Contributions

Daniel Severo did all the work.

References

Adams, R. P. and Zemel, R. S. Ranking via sinkhorn propagation. June 2011.

Burrows, M. and Wheeler, D. A block-sorting lossless data compression algorithm. In *Digital SRC Research Report*. Citeseer, 1994.

Cover, T. M. *Elements of information theory*. John Wiley & Sons, 1999.

Duda, J. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. November 2013.

Elias, P. Universal codeword sets and representations of the integers. *IEEE transactions on information theory*, 21(2): 194–203, 1975.

Frey, B. J. and Hinton, G. E. Free energy coding. In *Proceedings of Data Compression Conference - DCC '96*, pp. 73–81. ieeexplore.ieee.org, March 1996.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kattan, A. and Poli, R. Evolutionary lossless compression with gp-zip. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 1211–1218, 2008.

Kingma, F. H. *Improving Data Compression Based On Deep Learning*. PhD thesis, Erasmus University OF Rotterdam, 2019.

Kuhn, H. W. The hungarian method for the assignment problem, 1955.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

Parent, J. and Nowe, A. Evolving compression preprocessors with genetic programming. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pp. 861–867, 2002.

Radescu, R. Transform methods used in lossless compression of text files. *Romanian Journal of Information Science and Technology*, 12(1):101–115, 2009.

Robinson, A. H. and Cherry, C. Results of a prototype television bandwidth compression scheme. *Proceedings of the IEEE*, 55(3):356–364, 1967.

Ruan, Y., Ullrich, K., Severo, D., Townsend, J., Khisti, A., Doucet, A., Makhzani, A., and Maddison, C. J. Improving lossless compression rates via monte carlo Bits-Back coding. February 2021.

Ryabko, B. Y. Data compression by means of a "book stack". *Problemy Peredachi Informatsii*, 16(4):16–21, 1980.

Seward, J. bzip2 and libbzip2. *avaliable at http://www. bzip. org*, 1996.

Steinruecken, C. *Lossless data compression*. PhD thesis, University of Cambridge, 2015.

Townsend, J., Bird, T., and Barber, D. Practical lossless compression with latent variables using bits back coding. January 2019a.

Townsend, J., Bird, T., Kunze, J., and Barber, D. HiLLoC: Lossless image compression with hierarchical latent variable models. December 2019b.

Welch. A technique for High-Performance data compression. *Computer*, 17(6):8–19, June 1984.