



Università di Pisa

Reordering collections of texts for better compression

Pesaresi Seminars

Andrea Castagna

Lossless Compression

Formal definition of compression and decompression

Let Σ be a finite alphabet and let Σ^* be the set of all finite strings over Σ .

Formal definition

A **lossless compression scheme** consists of two mappings

$$C : \Sigma^* \rightarrow \{0, 1\}^*, \quad D : \{0, 1\}^* \rightarrow \Sigma^*$$

satisfying

$$D(C(x)) = x \quad \forall x \in \Sigma^*$$

Objective

The goal is to encode text exactly, but using fewer bits on average than the original representation.

$$\mathbb{E}[|C(x)|] < |x|$$

Why Lossless Compression Matters

Storage, communication, and exact reconstruction

Why study compression?

- **Storage:** reducing data size lowers storage costs at scale.
- **Transmission:** fewer bits enable faster communication over limited bandwidth.
- **Energy:** less data movement reduces energy consumption.

When lossless compression is needed

- **Exact reconstruction** is required
- Typical examples:
 - source code
 - databases and logs
 - medical or legal data

When lossy compression is acceptable

- **Small distortions** are tolerable
- Typical examples:
 - images (JPEG)
 - audio (MP3)
 - video (H.264)

What Do We Mean by Text?

Different textual domains exhibit different structure

Text

In compression, text is any symbolic sequence $S = s_1, \dots, s_n$ over a finite alphabet Σ .

Many kinds of symbolic sequences can be treated as text for compression purposes.

- **Natural language:** books, articles, web pages
- **Source code:** syntax-rich and highly repetitive
- **Structured data:** logs, JSON, XML
- **Biological sequences:** DNA over $\{A, C, G, T\}$

Key observation

These domains exhibit different statistical regularities. In this talk, we focus on **source code**, while many of the ideas extend more broadly to natural language texts.

Standard Text Compressors

Main approaches and their limitations

Typical approaches

- **Statistical coding:** Huffman coding, arithmetic coding
- **Dictionary-based:** gzip, zstd — replace repeated patterns with back-references
- **Block-sorting:** bzip2 — reorder text via BWT, then encode

Key limitation on text

Traditional compressors capture **local regularities**, but struggle with:

- long-range dependencies
- semantic and grammatical structure

- 1 BWT and Dictionary based compression
- 2 Proofs of computational intractability
- 3 Heuristics
- 4 Questions

Burrows-Wheeler Transform (BWT)

To compute the Burrows-Wheeler Transform (Burrows and Wheeler 1994):

- ▶ Generate a matrix of all the rotations of a string s .
- ▶ Order lexicographically the rotations in the matrix.
- ▶ Pick the last column of the matrix, $BWT(s)$.

$s = \text{abracadabra}$

										<i>BWT</i>
a	a	b	r	a	c	a	d	a	b	r
a	b	r	a	a	b	r	a	c	a	d
a	b	r	a	c	a	d	a	b	r	a
a	c	a	d	a	b	r	a	a	b	r
a	d	a	b	r	a	a	b	r	a	c
b	r	a	a	b	r	a	c	a	d	a
b	r	a	c	a	d	a	b	r	a	a
c	a	d	a	b	r	a	a	b	r	a
d	a	b	r	a	a	b	r	a	c	a
r	a	a	b	r	a	c	a	d	a	b
r	a	c	a	d	a	b	r	a	a	b

$BWT(s) = \text{rdarcaaabb}$

- ▶ Runs are maximal substring of identical characters in $BWT(s)$.
- ▶ Clustering effect: runs tend to form around rotations starting with repeated substrings.
- ▶ They are crucial for compression.
- ▶ We denote its number as r_{BWT}

$s = abracadabra$

											<i>BWT</i>
a	a	b	r	a	c	a	d	a	b		r
a	b	r	a	a	b	r	a	c	a		d
a	b	r	a	c	a	d	a	b	r		a
a	c	a	d	a	b	r	a	a	b		r
a	d	a	b	r	a	a	b	r	a		c
b	r	a	a	b	r	a	c	a	d		a
b	r	a	c	a	d	a	b	r	a		a
c	a	d	a	b	r	a	a	b	r		a
d	a	b	r	a	a	b	r	a	c		a
r	a	a	b	r	a	c	a	d	a		b
r	a	c	a	d	a	b	r	a	a		b

$$r_{BWT}(s) = 7$$

Dictionary-based compressors

- ▶ Search matches between the text they are compressing and a set of strings, often built from the text itself
 - ▶ Substitute each match with a pointer to the corresponding element in the set of strings.
-
- ▶ Sweep left to right the text, each substring of the processed text becomes part of the dictionary
 - ▶ Whenever you find a long enough match substitute the match with the position and length of the match in the processed prefix.

abracadabra $\rightarrow$ *abracad*(0, 4)

Sliding Window

$c_1 \dots c_i \dots \underbrace{c_j \dots c_{j+m}}_{\rightarrow} \dots c_k \dots c_n$

- ▶ gzip: 32 KB
- ▶ xz: 4 MB
- ▶ zstd: 2 MB

Blocks

$\underbrace{c_1 \dots c_m}_{\text{block 1}} \dots \underbrace{c_j \dots c_{j+m}}_{\text{block } i} \dots \underbrace{c_{n-m+1} \dots c_n}_{\text{block } k}$

- ▶ bzip2: 900 KB

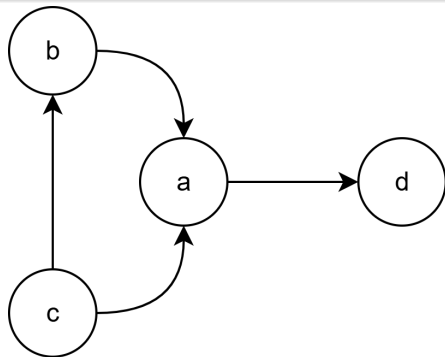
Objective

- ▶ A collection $S = \{T_1, \dots, T_n\}$ of texts of length t in alphabet Σ .
- ▶ The sliding compressor window of m characters spans up to $k = \lfloor \frac{m}{t} \rfloor + 1$ texts.
- ▶ For a permutation π , define consecutive windows $S_i = T_{\pi(i)} \dots T_{\pi(i+k-1)}$.
- ▶ Objective find:

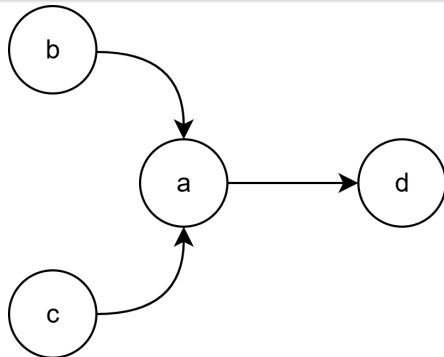
$$\pi^* = \arg \min_{\pi} \sum_{i=1}^{n-k+1} r_{BWT}(S_i),$$

Hamiltonian path problem

The Hamiltonian path problem asks whether or not in a graph we have a path that touches all vertices exactly once (an Hamiltonian path). It is a known NP-Complete problem. (Karp 1972)



Hamiltonian path $c \rightarrow b \rightarrow a \rightarrow d$.

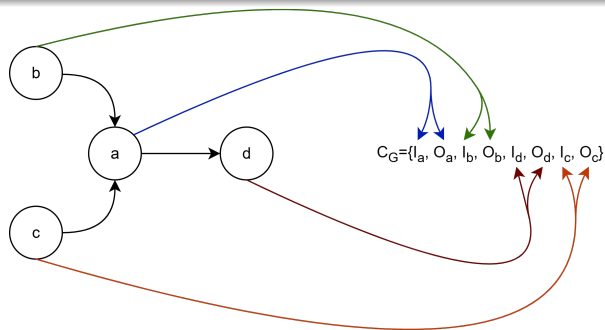


No Hamiltonian path.

High-level reduction idea

We reduce from Hamiltonian path problem:

- ▶ For a graph $G = (V, E)$ we define a set of texts $C_G = \{I_v, O_v\}_{v \in V}$.
- ▶ For each vertex v create two texts I_v and O_v that encode edges.
- ▶ The optimal ordering of texts C_G has a number of runs capable of distinguishing graphs with Hamiltonian paths.



Texts structure: I_v and O_v

I_v

$$I_v = \underbrace{\overbrace{h_v^1 h_v^2 h_v^{in}}^{\text{head}}}_{3} \underbrace{\overbrace{e_{w_1,v}^{in} e_{w_1,v}^{in} \dots e_{w_{indeg(v),v}}^{in} e_{w_{indeg(v),v}}^{in}}^{\text{incoming edges}}}_{2 \cdot indeg(v)} \underbrace{\overbrace{p_{v,1}^{in} \dots p_{v,2 \cdot (n-1-indeg(v))}^{in}}^{\text{padding}}}_{2 \cdot (n-1-indeg(v))}$$

$2n+1$

O_v

$$O_v = \underbrace{\overbrace{h_v^1 h_v^2 h_v^{out}}^{\text{head}}}_{3} \underbrace{\overbrace{e_{v,w_1}^{out} e_{v,w_1}^{out} \dots e_{v,w_{outdeg(v)}}^{out} e_{v,w_{outdeg(v)}}^{out}}^{\text{outgoing edges}}}_{2 \cdot outdeg(v)} \underbrace{\overbrace{p_{v,1}^{out} \dots p_{v,2 \cdot (n-1-outdeg(v))}^{out}}^{\text{padding}}}_{2 \cdot (n-1-outdeg(v))}$$

$2n+1$

Reduction statement

- ▶ If it exists an ordering $I_{v_1} O_{v_1} \dots I_{v_n} O_{v_n}$ or $O_{v_n} I_{v_n} \dots O_{v_1} I_{v_1}$ such that $(v_i, v_{i+1}) \in E \forall i \in [1, \dots, n-1]$ they are optimal and
- ▶ It exists a bijection between $I_{v_1} O_{v_1} \dots I_{v_n} O_{v_n}$ or $O_{v_n} I_{v_n} \dots O_{v_1} I_{v_1}$ and an Hamiltonian Path $v_1 \rightarrow \dots \rightarrow v_n$.

Theorem

The Hamiltonian Path problem reduces in polynomial time to the minimum total BWT-runs problem with window size $k = 2$. Hence the latter is NP-hard.

- ▶ Construction size: alphabet $\Theta(n^2)$, $2n$ texts of length $\Theta(n)$ each.
- ▶ Reduction runs in $\mathcal{O}(n^2)$ time.

Objective

- ▶ Assume writing a single character has cost 1 and a pointer cost c .
- ▶ Define $C(s_1|s_2)$ the cost of compressing s_1 assuming we have in the dictionary all the substrings of s_2 .
- ▶ For a permutation π , define consecutive texts window $P_i = T_{\pi(i-k+1)} \dots T_{\pi(i-1)}$, that represents the texts we have as prefix for compressing $T_{\pi(i)}$. (This can be empty)
- ▶ Objective find:

$$\pi^* = \arg \min_{\pi} \sum_{i=1}^n C(T_{\pi(i)}|P_i),$$

Texts structure: I_v and O_v

I_v

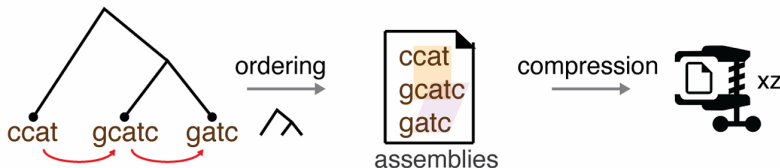
$$I_v = \underbrace{\overbrace{H_v}^{\text{head}} \underbrace{E_{w_1,v} \dots E_{w_{\text{indeg}(v)},v}}^{\text{incoming edges}} \underbrace{P_{v,1}^{\text{in}} \dots P_{v,2 \cdot (n-1-\text{indeg}(v))}^{\text{in}}}_{\text{padding}}}_{\substack{[\lceil c \rceil]+2 \\ (\lceil c \rceil+1) \cdot \text{indeg}(v) \\ (\lceil c \rceil+1) \cdot (n-1-\text{indeg}(v)) \\ (\lceil c \rceil+1)n+1}}$$

O_v

$$O_v = \underbrace{\overbrace{H_v}^{\text{head}} \underbrace{E_{v,w_1} \dots E_{v,w_{\text{outdeg}(v)}}}_{\text{outgoing edges}} \underbrace{P_{v,1}^{\text{out}} \dots P_{v,2 \cdot (n-1-\text{outdeg}(v))}^{\text{out}}}_{\text{padding}}}_{\substack{[\lceil c \rceil]+2 \\ (\lceil c \rceil+1) \cdot \text{outdeg}(v) \\ (\lceil c \rceil+1) \cdot (n-1-\text{outdeg}(v)) \\ (\lceil c \rceil+1)n+1}}$$

Phylgenetic compression (Brinda et al. 2024) consists of four main conceptual steps:

- ▶ Clustering/batching;
- ▶ Inference of a compressive phylogeny;
- ▶ Data reduction/reordering;
- ▶ Compression using a low level compressor.



Phylogenetic Compression

Collection	Phylogenetic compression protocol			Compression statistics				Improvement over baselines		
	Reordering scheme	Compressive phylogeny	Low-level compressor	Compressed size	Kilobytes per genome	Bits per bp	Bits per distinct k -mer	FASTA-like baseline (8 bits per bp)	GZip-like baseline (2 bits per bp)	Original compression protocol
GISP	Left-to-right	Mashtree	XZ	5.67 MB	5.15	0.0192	10.9	416.×	104.×	-
	Left-to-right	RAxML	XZ	5.44 MB	4.94	0.0184	10.4	434.×	109.×	-
NCTC3k	Left-to-right	Mashtree	XZ	257 MB	242	0.473	2.07	16.9×	4.23.×	-
SC2	Left-to-right	Sarscov2phylo	XZ	10.7 MB	0.0181	0.00486	46.2	1,647×	412×	-
BIGSIdata	Batches & k -mer propagation	Mashtree (1 tree/batch, 574 batches)	XZ	52.3 GB	123.	0.248	10.2	32.2×	8.06×	319×
661k	Batches & left-to-right	Mashtree (1 tree/batch,	XZ	29.0 GB	43.8	0.0898	5.23	89.1×	22.3×	27.8×
			MBGC	20.7 GB	31.3	0.0642	3.74	125×	31.2×	38.9×
	Lexicographically by ENA accessions ^b	-	XZ	121 GB	182.	0.374	21.8	21.4×	5.35×	6.67×
			MBGC ^c	-	-	-	-	-	-	-

Table: Results of experiments in Efficient and Robust Search of Microbial Genomes via Phylogenetic Compression (Brinda et al. 2024)

In On the compressibility of large-scale source code datasets (Boffa et al. 2025) compared effectiveness of different frameworks of data reordering on compressibility:

- ▶ git-pack;
- ▶ disk-based construction bwt (Ferragina et al. 2012);
- ▶ permute partition compress schema (Buchsbaum et al., 2000, 2003):
 - ssdeep Kornblum (2006);
 - TLSH Oliver et al. (2013);
 - SimHash Charikar (2002);
 - MinHash. Broder (1997);
 - filename;
 - filepath.

On the compressibility of large-scale source code datasets

Technique	Compression ratio (%)	Compression speed (MiB/s)	Decompression speed (MiB/s)
single_compress + gzip	32.4	11.4	65.5
single_compress + zstd-3	33.4	79.5	102.9
single_compress + zstd-22	29.9	16.7	117.7
random_order + zstd-22	18.2	2.4	304.6
git-pack, $w = 10, d = 50$	20.9	49.2	54.4
git-pack, $w = 20, d = 100$	20.2	36.7	56.9
git-pack, $w = 40, d = 200$	19.9	28.3	66.9
bwt_rle	12.0	<1	<1
csa_sada, $s = 128$	34.8	3.6	3.7
csa_sada, $s = 256$	31.3	2.6	2.9
csa_wt, $b = 63$	25.8	2.2	<1
csa_wt, $b = 127$	23.4	1.8	<1
tlsh + zstd-22	10.5	12.4	329.6
ssdeep + zstd-22	10.7	13.0	331.7
simhash + zstd-22	10.8	11.0	330.7
minhash_graph + zstd-22	10.3	3.3	332.1
type_minhash_graph + zstd-22	9.5	1.7	327.4
filename + zstd-22	10.1	32.9	379.7
path + zstd-3	15.8	465.8	326.3
path + zstd-22	10.1	32.7	381.0

Table: Results of experiments in On the compressibility of large-scale source code datasets (Boffa et al. 2025)

Open Questions

- ▶ Is the problem also intractable for finite alphabets? What about block schemes?
- ▶ Can we use general optimization techniques for finding approximate solutions? Could it be possible to find algorithms with bounded error?
- ▶ How can we use machine learning techniques to aid in the task of clustering similar texts?