Universidad de Concepción
Dirección de Postgrado
Facultad de Ingeniería - Programa de Doctorado en Ciencias de la Computación

COMPACT DATA STRUCTURES FOR RASTER DATA

Tesis para optar al grado de
DOCTOR EN CIENCIAS DE LA COMPUTACIÓN

POR
Martita Paulina Muñoz Candia
CONCEPCIÓN, CHILE
2025

Profesor guía: Cecilia Hernández Rivas
Profesor co-guía: José Fuentes Sepúlveda
Profesor guía externo: Diego Seco Naveiras
Departamento de Ingeniería Informática y Ciencias de la Computación
Facultad de Ingeniería
Universidad de Concepción

A mi mamá, quien me ha apoyado toda la vida.

AGRADECIMIENTOS

Quisiera comenzar agradeciendo a Dios por permitir lograr esta oportunidad de poder estudiar este doctorado en esta universidad, y lograr llegar hasta esta instancia de la vida.

Quiero agradecer a mi mamá, quien siempre me ha apoyado en todos los momentos claves de mi vida, y a mi novio, quien siempre ha estado a mi lado con todo su apoyo y cariño. Ellos han sido mi pilar fundamental en todo este largo proceso.

También agradecer a mis directores de tesis. Realmente aprendí mucho trabajando con la profesora Cecilia, el profesó José y el profesor Diego. Ellos fueron un aporte trascendental durante el desarrollo de esta tesis, quienes me ayudaron y guiaron en este largo, complejo e increíble camino.

Han sido varios años de trabajo, en una tesis que por momentos me ha parecido interminable, llena de experiencias que jamás imaginé vivir. Cada uno de estos momentos ha quedado marcado en mis recuerdo, en mi experiencia y en mi corazón. Gracias a la universidad, y a todos quienes me han ayudado a hacer posible este sueño.

Abstract

A raster model consists of a matrix of cells in which each cell contains a value that represents information. A raster time series is an ordered collection of independent rasters. Raster and raster time series models have been applied in different domains, such as terrain altitude, temperature, atmospheric pressure, images, etc. Usually, the raster time series represents the raster data changes over time.

The main attribute of those models is the data locality. Data locality indicates that contiguous cells (spatially or temporally) contain similar values. Compression data exploits this characteristic. Besides, compact data structures allow the data to be compacted and respond to different queries without descompacting. Compact data structures have been used to represent raster data and raster time series, but it is possible to improve the efficiency of these structures.

The work on this thesis focuses on improving the raster and raster time series compression. The thesis introduces an algorithm for constructing the Heuristic T-k^2-raster that integrates a dynamic-programming approach, and proposes a new compact data structure for raster time series, the ZT-k^2-raster, based on the k^2-raster model. Additionally, alternative heuristic construction algorithms that incorporate clustering techniques have been developed to exploit temporal patterns better. The work also includes the implementation of a massively parallel k^2-raster construction algorithm for GPGPU architectures, together with a comprehensive experimental evaluation of all proposed methods and their performance.

The experimental results confirm that these strategies improve compression and representation under specific conditions. The dynamic-programming approach consistently outperforms the baseline heuristic on synthetic datasets with high temporal and low spatial locality. In contrast, the clustering-based heuristics provide clear advantages for datasets with cyclic temporal patterns. The analysis of compressibility measures reveals a strong positive correlation with raster size and a strong negative correlation with spatial locality. Furthermore, a new alphabet-sensitive 2D compressibility measure is introduced to better characterize raster compressibility. Finally, ZT-k^2-raster achieves a competitive space–time trade-off for representing raster time series, and the parallel implementation demonstrates substantial acceleration during the construction phase.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

A raster model consists of a matrix of cells in which each cell contains a value that represents information. Raster models can represent diverse phenomena, such as terrain altitude, temperature, atmospheric pressure, or digital images. A raster time series is an ordered collection of independent rasters. Its main application is modeling the temporal evolution of the phenomenon represented in each raster.

Raster data (and a raster time series) are often collected by geostationary satellites, which generate a large volume of information at a high frequency and resolution [152]. Advances in remote sensing and instrumentation have accelerated the growth of raster datasets [98]. For example, estimates indicate that remotely sensed imagery amounts to several terabytes acquired daily [103], and the accumulated archive of raster data is approaching the zettabyte scale [103].

Two factors determine the storage space required by a raster. The first factor is the *spatial resolution*, which depends on cell size: reducing cell size increases precision and the space consumption. The second factor is the *temporal resolution*, which depends on the interval between consecutive rasters in a raster time series: shorter intervals increase temporal precision and the space consumption. Traditionally, various formats and tools exist to store raster data. Some formats are an array of values or a raster-based image file format [153]. For example, GeoTIFF[1] builds on conventional image formats while adding metadata to describe geographic attributes. NetCDF [101] enables the storage and sharing of scientific data in matrix form, integrating Deflate-based compression internally [47].

Raster time series have typically followed the same storage strategies as individual

[1] `https://trac.osgeo.org/geotiff/`

rasters. One approach uses a generic multidimensional array representation, while another uses classic techniques for representing image sequences (e.g., video formats). For example, NetCDF represents raster time series due to its multidimensional data compression capabilities [101].

Raster data also presents a property known as data locality. Adjacent cells, spatially or temporally, tend to hold similar or slightly varying values. This property can be exploited for data compression, as in the case of Compact Data Structure (CDS) [76]. CDSs are data structures that seek to represent data succinctly or compactly, enabling query support without decompacting.

The work on this thesis focuses on improving the space/time Compact Data Structures for representing and querying raster data and raster time series. This work explores parallelism, compact indexing, and compressibility measures. Furthermore, it examines the application of CDS to raster data and raster time series in domains such as scientific images and related fields.

1.2 Hypothesis and goals

1.2.1 Hypothesis

This thesis is based on the following hypothesis: *It is possible to improve the compression and efficiency of algorithms on CDS applied in raster data using parallelism and compact indexes.*

We complement this hypothesis with the following research questions:

- I) What are the limitations of the existing CDS that represent raster data and raster time series?
- II) How can we design efficient parallel algorithms for constructing CDS for raster data?
- III) What are the most effective compressibility measures for achieving the optimal compression of raster data?
- IV) How can CDS be optimized for raster data to improve query performance in scientific images?

1.2.2 Main goal

The main goal of this thesis is to explore raster and raster time series compression, applying methods such as compact indexes, clustering, compression measures, and parallel computing.

1.2.3 Specific goals

1. To improve compression of raster time series indexed on the T-k^2-raster applying dynamic programming and clustering methods.
2. To evaluate raster compression via compressibility measures and propose a new compressibility measure applied to rasters.
3. To design and implement parallel algorithms for constructing CDS for raster data.
4. To analyze raster compression and query performance improvements on scientific images.

1.3 Methodology

Each specific goal includes one or more proposals evaluated through a structured experimental process comprising four steps: design, theoretical analysis, implementation, and experimentation. Each step was adapted according to the specific goal considered.

For the design step, we consider the following:

- We designed sequential algorithms following the classical Von Neumann architecture, where data reside in main memory (RAM).
- We designed parallel algorithms using Massive Parallel Computation on Multicore in a Graphics Processing Unit (GPU).

For the theoretical analysis, we consider the following:

- We analyzed sequential algorithms under the word random access model (RAM) model, defining their temporal and spatial complexity using Big-O notation.

- We analyzed parallel algorithms using the parallel random access model (PRAM), characterizing their work, depth, speedup, and efficiency.
- We analyzed data structures, regarding their compression ratio relative to the uncompressed data.

For the implementation step, we consider the following:

- We implemented all algorithms and data structures in C/C++, employing appropriate tools and compilers for parallel computation.

For the experimentation step, we consider the following:

- We compared all algorithms and data structures with the state-of-the-art approaches. The implementation of k^2-raster is available in a public repository[2].
- We used real-world datasets whenever possible, especially those employed in previous studies from the state-of-the-art. One of the considered datasets corresponds to the work described in [98] (Section 5.2), and it is available in a public repository[3]. For the fourth specific goal, we included datasets from [113] that describes medical images representing different protein from a mouse liver tissue cells.
- We consider two measures for algorithm benchmarking: time and memory usage. We measure the memory usage by the structure and the operation time (including construction) for data structures. We consider the `<chrono>` library from C++ for time measurement.

1.4 Contributions

After completing this thesis, the main contributions can be summarized as follows:

- Development of an algorithm for constructing Heuristic T-k^2-raster [33, 153] that integrates a dynamic programming approach (Chapter 3).

[2] `https://gitlab.lbd.org.es/fsilva/k2-raster`

[3] Rasters: `https://zenodo.org/records/17594578`, Temporal Rasters: `https://zenodo.org/records/17590893`

- Design of a Compact Data Structure (CDS) based on the k^2-raster [97, 98] to represent raster time series, referred to as the ZTk^2-raster (Chapter 3).
- Development of alternative heuristic algorithms for T-k^2-raster construction that incorporate clustering-based techniques (Chapter 4).
- A comprehensive experimental evaluation of all proposed algorithms and their performance.
- An extensive analysis of different compressibility measures applied to raster data and raster time series (Chapter 5).
- Definition of a new alphabet-sensitive compressibility measure for raster data (Chapter 5).
- An implementation of an algorithm for k^2-raster construction using massively parallel computation on GPGPU architectures (Chapter 6).

The work has resulted in several publications derived from this thesis:

- Martita Muñoz, Jose Fuentes-Sepúlveda, Cecilia Hernandez, Gonzalo Navarro, Diego Seco and Fernando Silva-Coira. Clustering-based compression for raster time series. The Computer Journal, 68(1): 32-46, 2024 (Chapter 4) [118].
- Martita Muñoz, Jose Fuentes-Sepúlveda, Cecilia Hernandez, and Diego Seco. Estimating the compressibility of raster data. Information Systems, 136: 102624, 2025 (Chapter 5) [119].

These contributions have been presented through participation in the following events:

- Muñoz, M., Hernández C., Seco D., Fuentes J.: Parallel Construction of k^2-raster using GPGPU. XIV Workshop Centre of Biotechnology and Bioengineering (2021).
- Muñoz, M., Hernández C., Seco D., Fuentes J.: Succinct representation of medical images. XV Workshop Centre of Biotechnology and Bioengineering (2022).

- Muñoz, M., Hernández C., Seco D., Fuentes-Sepúlveda J., Navarro G., Silva-Coira F.: Clustering based compression for raster time series. 18th Workshop on compression, Text, and Algorithms (WCTA) (2023).
- Muñoz, M., Hernández C., Seco D., Fuentes-Sepúlveda J., Navarro G., Silva-Coira F.: Clustering based compression for raster time series. Workshop "Estructuras de datos compactas" Universidad del Bío-Bío (2024).

1.5 Contents

This report is organized into the following chapters: Chapter 2 introduces the preliminary concepts relevant in this thesis. Chapter 3 presents two proposals for enhancing raster time series compression. Chapter 4 introduces an improvement to the T-k^2-raster heuristic by applying clustering techniques. Chapter 5 evaluates different compressibility measures applied to raster data. Chapter 6 describes our proposal for constructing the k^2-raster using massively parallel computation on GPGPU architectures. Finally, Chapter 7 summarizes the principal conclusions of this thesis and presents the future work.

Chapter 2

Preliminars

This chapter presents the theoretical background relevant to understanding the development of this thesis. Section 2.1 introduces Compact Data Structures (CDS), highlighting the k^2-tree, the structure on which the main data structures employed in this thesis are based. Section 2.2 describes the CDSs presented in the literature to represent raster data, highlighting the k^2-raster. Section 2.3 reviews the CDS presented in the literature to represent raster time series, highlighting the Heuristic T-k^2-raster. Finally, Section 2.4 presents the applications of the CDS used to represent rasters and raster time series described in the literature.

2.1 Compact Data Structures

A *Compact Data Structure* is a data structure that aims to represent different types of data using the least amount of space possible, supporting queries without decompressing the data [76]. A CDS is different from a compressor because the latter require fully or partially decompressing the data to perform queries.

A relevant CDS is the *bitmap*, which is a binary array that efficiently answers two relevant queries. The first query is the $\texttt{rank}_x(i)$ query, which counts the number of symbols x among the positions $[1, i]$ of the bitmap. The second query is the $\texttt{select}_x(i)$ query, which returns the position within the array where the i-th occurrence of x appears.

Figure 2.1 presents an example of both queries on a bitmap. The query $\texttt{rank}_1(8) = 3$ returns the number of 1's that appear between position 1 and 8. The query $\texttt{select}_1(3) = 4$ returns the position where the third one appears.

Several proposals implement a bitmap efficiently [76, 40, 116, 133, 138, 130, 124]. The most recent proposals suggest that precomputing the queries for some values and storing these results in an additional structure. The query on non-precomputed values

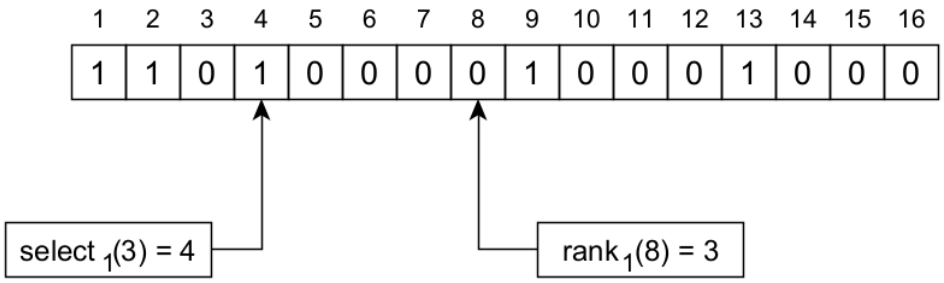


Figure 2.1: Example of a bitmap with the rank and select query examples.

utilizes the precomputed answers. This proposal allows us to answer both queries in $O(1)$ time using $o(n)$ bits of additional space.

2.1.1 k^2-tree

The k^2-tree is a CDS designed to represent sparse binary matrices [26, 28, 96]. This structure is the base for developing different CDSs to represent raster data. Its initial goal was to represent Web graphs indexed in an adjacency matrix. The k^2-tree is based on the strategy used in the Quadtree [144]. This structure reduces space consumption by compressing zero-valued regions, achieving better compression on matrices with clustered zeros.

Let a square matrix of size $n \times n$, with n power of k. If n is not a power of k, the matrix can be extended to the smallest value $n' > n$, which is a power of k. The building process subdivides recursively the matrix into k^2 submatrices. Next, it represents the subdivision with a conceptual tree. Each submatrix is represented with a zero if it contains only zeros or a one if it contains at least one 1. In case a submatrix contains only zeros, the subdivision terminates. Two bitmaps represent the conceptual tree: T and L. The building process traverses the tree level by level, storing all levels of the tree in those bitmaps: all levels except the last level in T and the last level in L.

Figure 2.2 presents an example of a k^2-tree over a 8×8 binary matrix with $k = 2$. The lower right matrix contains only zeros. The structure does not subdivide the matrix or extend this node in the conceptual tree. Therefore, it represents a 4×4 submatrix with a single bit set to 0. The process subdivides the remaining submatrices, saving

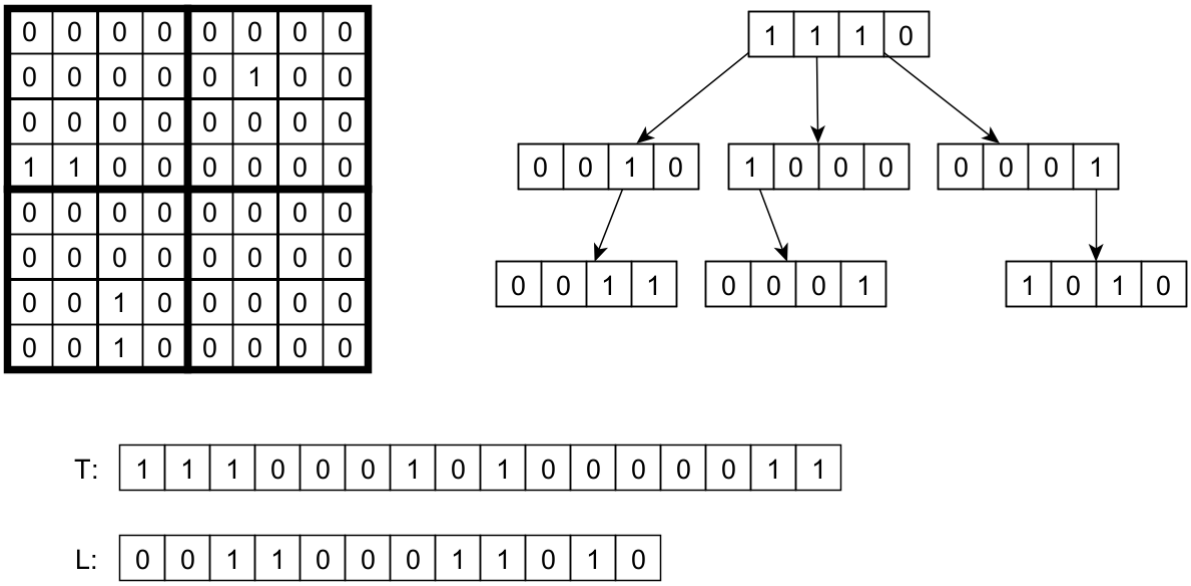


Figure 2.2: Complete example of a k^2-tree on a raster of size 8×8, with its conceptual tree and bitmaps

space in the 2×2 submatrices that contain only zeros.

To perform queries, the structure is traversed from the root to the leaves by iterating over the arrays T and L. The i-th child of a node at position x in T is obtained using the operation $\texttt{child}_i(x) = \texttt{rank}_1(x) \times k^2 + i$. If the resulting position exceeds the size of T (i.e., $\texttt{child}_i(x) > |T|$), the child corresponds to a cell represented in L at position $\texttt{child}_i(x) - |T|$.

2.2 CDS on raster data

There are different CDS proposals to represent raster data. In addition to compacting the raster, the CDSs seek to answer Map Algebra queries. Within these queries, there are three that are the most relevant: (1) `getCell` obtains the value of a cell located at position (r, c), (2) `getWindow` obtains the values within a query window $[r_1, r_2] \times [c_1, c_2]$, and (3) `getWindowRange` obtains the positions within a query window $[r_1, r_2] \times [c_1, c_2]$ that contain values within a range $[v_1, v_2]$.

De Bernardo *et al.* introduced the first CDSs based on a k^2-tree to represent raster data: k^2-base, k^2-acc, and k^3-tree [44]. Let σ be the set of distinct values within a raster.

k^2-base uses a k^2-tree for each value $v \in \sigma$ to represent all cells where v appears. k^2-acc uses a k^2-tree for each value $v \in \sigma$ to represent all locations where values are less or equal than v. k^3-tree is an extension of k^2-tree, which represents each cell as a three-dimensional tuple $\langle r, c, v \rangle$, where r and c represent the cell position and v represents its value. Spatial subdivision is performed similarly to the k^2-tree.

The k^2-base and k^2-acc are structures that efficiently answer the `getWindowRange` query. The k^2-base requires access to as many k^2-trees as the size of the range $[v_1, v_2]$, while the k^2-acc only requires access to two k^2-trees to answer the query. On the other hand, the k^3-tree is a structure that efficiently answers the `getCell` query.

The k^2-acc includes different extensions of its functionalities. Brisaboa *et al*. presented a method to perform queries between spatial objects (points, lines, polygons) indexed in an R-tree and a raster indexed in a k^2-acc [22]. Caniupán *et al*. presented the development of different Map Algebra queries on the k^2-acc [30].

Brisaboa *et al*. proposed three CDSs to represent raster data [23]. These structures are based on the k^2-ones, a variant of the k^2-tree that compresses submatrices full of zeros or ones [23]. The k^2-ones incorporates an additional bitmap into its representation, determining whether each unexpanded submatrix contains only zeros or only ones. A second related CDS is the Ik^2-tree [9]. The Ik^2-tree is a CDS based on the k^2-tree and designed to represent ternary relations, i.e., rasters where some cells contain an integer value. The structure represents each node using at most $|\sigma|$ bits, where $|\sigma|$ is the number of different values inside the raster. These bits indicate which values are present within the corresponding submatrix. The structure compresses submatrices that contain no values.

The structures proposed by Brisaboa *et al* are [23]: Multiple k^2-ones or Mk^2-ones, Cumulative k^2-ones or Ck^2-ones and Interleaved k^2-ones or Ik^2-ones. The Mk^2-ones is based on the idea of the k^2-base, replacing the k^2-tree with the k^2-ones. The Ck^2-ones is based on the idea of the k^2-acc, replacing the k^2-tree with the k^2-ones. The Ik^2-ones combines the k^2-ones with the Ik^2-tree. Mk^2-ones and Ck^2-ones are structures that efficiently answer the `getWindowRange` query, while Ik^2-ones allow compressing rasters with large areas with similar values efficiently. As De Bernardo *et al*. proposed, the main limitation of these structures is the efficient processing of rasters with many

different values.

Brisaboa *et al*. proposed two CDSs to represent rasters, but they focused on answering the *Top-*K query [23]. This query returns the positions of the cells with the highest or lowest values in the raster. These structures are based on the k^2-treap [24]. The two proposed structures are: the k^2-treap-uniform or k^2-treap^U, and the k^2-treap-uniform-or-empty or k^2-treap^{UoE}. The k^2-treap^U offers higher compression, while the k^2-treap^{UoE} is more efficient in queries.

Pinto *et al*. proposed two CDSs to represent raster data [136]. Their structures' strategy is linearizing the raster by applying the Morton Curve or Z-Order Curve [115], transforming the 2D raster into a 1D sequence. The proposed structures are: 2D1D-map and 3D2D-map. The 2D1D-map indexes the generated sequence in a *Differentially Encoded Search Tree* (DEST) [42]. The 3D2D-map represents each sequence cell in a binary matrix $X \times Y$. The X axis represents the cell position, while the Y axis represents the cell value. Next, store the resulting matrix in a k^2-tree. Both structures are competitive in terms of compression space and query time. The 3D2D map is the most efficient for `getWindow` and `getWindowRange` queries when the range is extensive.

There are two recent proposals in the literature on CDS applied to rasters. Pereira and Kaster presented a CDS to represent raster data called Compressed Line Raster or CL-Raster [53]. Their proposal uses a row-by-row linearization to subsequently compress the linearized sequence using Run-Length Encoding [69]. CL-Raster is efficient for sums and average values inside windows. Brisaboa *et al*. proposed a CDS based on the Block Tree [20]. The Two-Dimensional Block Tree or 2D-BT [25] is a structure that seeks to compress by exploiting repeated patterns within the raster.

Ladra *et al*. proposed a CDS called k^2-raster [97, 98]. Among the existing proposals in the literature, the k^2-raster offers the best trade-off between between compression efficiency and query performance. For this reason, we describe it in more detail below.

Table 2.1 presents a summary of the CDS designed to represent raster data.

2.2.1 k^2-raster

This k^2-raster follows a similar strategy to the k^2-tree, subdividing the raster into k^2 subrasters recursively. The structure stores each subraster's minimum and maximum

Structure	Based on	Observation	References
k^2-base	k^2-tree [26, 28, 96]	Efficient on *getWindowRange* query	[44]
k^2-acc	k^2-tree [26, 28, 96]	Efficient on *getWindowRange* query	[44]
k^3-tree	k^2-tree [26, 28, 96]	Efficient on *getCell* query	[44, 23]
Mk^2-ones	k^2-ones	Similar to k^2-base	[23]
CMk^2-ones	k^2-ones	Similar to k^2-acc	[23]
Ik^2-ones	Ik^2-tree [9]	Efficient compression in zones with similar values	[23]
2D1D map	Morton Curve [115]	Competitive on space-time	[136]
3D2D map	Morton Curve [115], k^2-tree [26, 28, 96]	Efficient on *getCell* query	[136]
2D-Block Tree	Block Tree [20]	Compress repeated patterns	[25]
CL-raster	Row-Major Curve, Run-Length Encoding [69]	Recent proposal	[53]
k^2-raster	k^2-tree [26, 28, 96]	Scalable for large volumes of data	[97, 98]

Table 2.1: Compact Data Structures (CDS) for representing raster datas presented in the literature

values and stops the subdivision at matrices that contain the same value in all their cells. It then traverses the resulting tree by levels and stores the values of each node and the structure of the tree.

It uses two arrays indexed into a Directly Addressable Code (DAC) vector [27]. The `Lmax` array stores the maximum values of each node, and the `Lmin` array stores the minimum values of each node. On the other hand, the structure uses two variables, `rMax` and `rMin`, to store the root's minimum and maximum. Additionally, it uses the bitmap T of the k^2-tree to store the tree structure. An additional strategy applied to improve compression is to represent each value within the nodes as the difference from the value of its parent node.

Figure 2.3 presents a complete example of a k^2-raster representation. On the top, the figure shows an 8×8 raster example and its respective conceptual tree representation using $k = 2$. On the bottom, it is shown the conceptual tree of differences and the final components of the data structure. The structure recursively subdivides the matrix into four submatrices, and each generated submatrix's minimum and maximum values

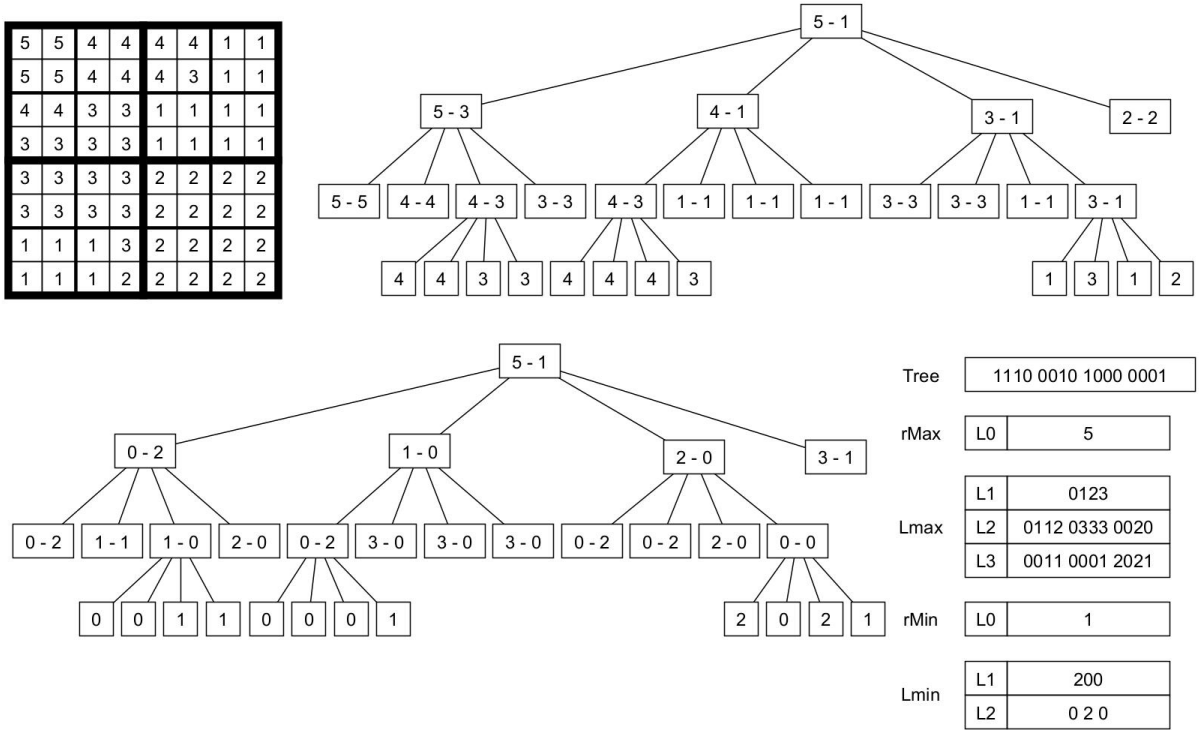


Figure 2.3: Complete example of a k^2-raster on a raster of size 8×8, with its conceptual tree and structures

are represented in the tree. If a submatrix contains only equal values, the subdivision process terminates for that submatrix, as seen in the lower right submatrix of the example, where all the cells contains the value 2. However, for the remaining submatrices, the recursive subdivision continues.

The next step is to compute the differences between the values of the nodes and their parent's nodes values. As a result, it returns the conceptual tree representation presented on the bottom left part of the figure. The root node stores the global maximum and minimum values of 5 and 1, respectively. The first submatrix in the top left corner contains 5 and 3, corresponding to the range of values inside the submatrix. The difference between the maximum value of the submatrix and that of its parent node is 0, whereas the difference between the respective minimum values is 2. Therefore, the submatrix stores the values of 0 and 2 as its maximum and minimum values, respectively, using this approach.

The structure is the most efficient presented in the literature, performing better data

compression and query times. Furthermore, this structure maintains its efficiency for large data sets containing large alphabet sizes.

Ladra *et al*. further proposed two variants. The first is a hybrid version called k^2-raster$_H$ [97]. In this variant, instead of using a single k value for recursive subdivision, the structure uses two k values: a k_1 that subdivides the first n_1 levels and a k_2 that subdivides the rest of the levels. The results show similar performance to the original k^2-raster but with improved `getCell` query time.

The second variant is an entropy-based heuristic version called k^2_H-raster [98]. The structure aims to compress the last level of the tree, composed of submatrices of size $k_{Lst} \times k_{Lst}$, representing the values of the original matrix. The proposal uses a dictionary associating each submatrix with a variable-length code, giving the shortest code to the most frequent submatrices. With this dictionary, the structure replaces the submatrices with the generated codes. The results show that k^2_H-raster improves compression and query efficiency.

The literature presents different extensions of the k^2-raster functionalities. Different investigations explore Map Algebra queries over this structure [45, 152]. Silva-Coira *et al*. presented a method to perform queries between spatial objects (points, lines, polygons) indexed in an R-tree and a raster indexed in a k^2-raster [154]. Additionally, it includes spatial join queries between the spatial objects and the raster with restrictions on the values within the raster. For example, it searches for the K spatial objects that overlap the cells with the raster's highest (or lowest) values.

2.3 CDS on raster time series

De Bernardo *et al*. proposed an extension of the k^2-tree to represent a raster time series [44]. The k^4-tree is a CDS that allows representing binary data in 4 dimensions. The first two dimensions represent the spatial position of each cell, the third represents the temporal location, and the fourth represents the value.

Pinto *et al*. proposed representing each raster of the raster time series in a 3D2D map and storing the generated binary matrices in a k^3-tree [136]. Cruces *et al*. implemented this idea and presented the 4D3D map [43]. Although the structure presents competitive results against window and range queries, it is inefficient compared to more

recent proposals.

Silva-Coira *et al*. proposed a CDS called T-k^2-raster [33, 153], based on the k^2-raster. Among the existing proposals in the literature, the T-k^2-raster offers the best trade-off between between compression efficiency and query performance. For this reason, we describe it in more detail below.

Table 2.2 presents a summary of the CDS designed to represent temporal raster data.

Structure	Based on	Observation	References
k^4-tree	k^n-tree	Naive proposal	[44]
4D3D mapping	3D2D mapping [136]	Competitive in queries	[43]
T-k^2-raster	k^2-raster [97, 98]	Best proposal on literature	[33, 153]

Table 2.2: Compact Data Structures (CDS) for representing raster time series presented in the literature

2.3.1 T-k^2-raster

This structure classifies each raster into two categories: `snapshot` and `log`. Snapshots appear in every fixed number of rasters' timestamps, while the rest are log rasters. The structure represents each snapshot into a k^2-raster (or a k_H^2-raster) and each log into a modified k^2-raster version called k^2-raster$_{log}$. The k^2-raster$_{log}$ stores the values of each node as the difference to the equivalent node of the last generated snapshot raster. Therefore, in addition to stopping subdivisions into submatrices with equal values, the k^2-raster$_{log}$ also stops subdivisions into submatrices whose difference to their snapshot is a constant value. The T-k^2-raster strategy captures spatial locality within each snapshot through its k^2-raster representation, and temporal locality through the differential encoding between the snapshot and the log rasters.

Figure 2.4 presents an example of a T-k^2-raster with a raster snapshot and a raster log. At the top, we can observe the raster snapshot and its respective k^2-raster. At the bottom, we can observe the raster log with its respective k^2-raster$_{log}$. The maximum and minimum values within the entire raster log are 9 and 3, respectively. The differences in these values compared to the raster snapshot are 0 and 1, respectively.

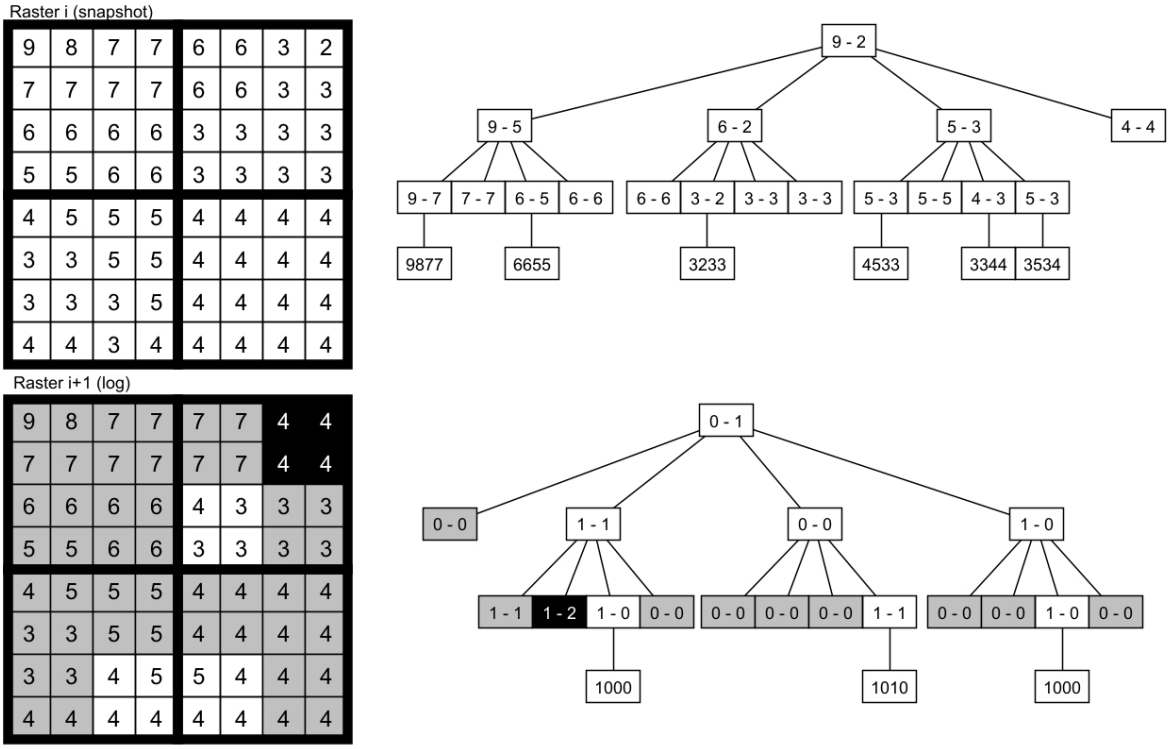


Figure 2.4: Complete example of a T-k^2-raster, with a snapshot raster and its respective raster log. The gray and black submatrices/nodes in the second raster represent the two cases that the k^2-raster$_{log}$ stops the subdivision. Black submatrices/nodes reflect the standard case of the k^2-raster when all the values in the submatrix are equal. Gray submatrices/nodes represent when the values between the submatrix and its equivalent in the snapshot vary by a constant.

Finally, the k^2-raster$_{log}$ stores those values. The gray submatrices and nodes of the raster log are not expanded because their values compared to the snapshot are constant. The black submatrices and nodes are those are not expanded because all their values inside the raster are equal. This case comes from the original condition of the k^2-raster.

The structure includes a new `eqB` bitmap to distinguish why the k^2-raster$_{log}$ stops the subdivision. When the value of `eqB` is 0, the subdivision stops because all its values are equal. When the value of `eqB` is 1, the subdivision stops because the difference with the snapshot is constant.

Silva-Coira *et al.* proposed a variant of the T-k^2-raster called *Heuristic T-k^2-raster* or T_H-k^2-raster [153]. This variant proposes a heuristic that selects raster snapshots,

selecting those that improve the compression of the raster time series. The heuristic reviews each raster timestamp in temporal order and decides to represent it using the least space. The three alternatives it presents are the following: (1) represent the raster as a snapshot, (2) represent the raster as a raster log concerning the last generated snapshot, or (3) represent the previous raster as a snapshot and the current raster as a log concerning the previous raster. Because snapshot rasters now do not appear at a fixed interval, the structure includes a bitmap called $\mathtt{sB}$ that determines which rasters are snapshots. When sB$[i] = 1$, the raster is a snapshot. When $\mathtt{sB}$[i]=0, the raster is a log. A raster log can find its previous raster snapshot using a rank query on $\mathtt{sB}$.

The proposed heuristic has certain limitations. The first is that the heuristic restricts the inclusion in the snapshot set of rasters that happened before the previous raster in revision, and the heuristic does not allow for removing rasters from the snapshot set. The second limitation is the selection of logs that a snapshot can reference. Specifically, exclusively the most recently generated snapshot in the temporal order can reference a log. This constraint may limit the ability of the heuristic to identify the optimal selection of snapshots to improve the compression. These limitations are addressed in the proposals described in Chapters 3 and 4.

2.4 CDSs for raster data: applications

The application of CDS on raster data includes different domains. The main domain is *Geographic Information System*s (GIS) [54]. These systems handle data on natural phenomena. GIS uses rasters and raster time series to model information on temperature, atmospheric pressure, wind speed, precipitation, and terrain elevation, among others. Rasters represent this information over a delimited space, while raster time series represent it over a delimited space and time range. Geostationary Satellites are responsible for producing large volumes of rasters. The CDS presented in this section were designed for this domain [44, 23, 136, 98, 33].

Another relevant domain is hyperspectral images, which contain multiple bands across the electromagnetic spectrum. The data is pulled from certain bands in the spectrum to help us find the objects we are specifically looking for, such as oil fields and minerals [35]. Airborne sensors and hyperspectral satellites are the ones that collect

this type of imagery. Chown *et al*. studied the compact representation of hyperspectral images using the k^2-raster [35, 38, 37, 36].

Raster data is beneficial for image compression [137, 74]. The main areas where image compression is relevant are medicine [145, 104] and astronomy [135, 106, 168, 105]. In both cases, noise in images poses a challenge when manipulating them [4].

Chapter 3

Compact Data Structures for Enhanced Raster Compression

3.1 Introduction

This chapter focuses on enhancing raster time series compression by designing and optimizing Compact Data Structures (CDS). We present two approaches to improve raster time series compression. The first proposal aims to evaluate the proposed heuristic of the Heuristic T-k^2-raster [33, 153], comparing it with Dynamic Programming (DP). DP is a programming strategy that guarantees optimal raster snapshot selection. With this evaluation, we aspire to determine the optimality of the heuristic and its eventual improvements to reduce the compaction size of a raster time series.

The second proposal is a new CDS representing raster time series: ZT-k^2-raster. This technique linearizes each raster using Morton Curve or Z-Order curve [115] and then generates a two-dimensional raster by representing each raster from the temporal raster as a row. Finally, ZT-k^2-raster represents the generated raster by a k^2-raster [97, 98]. This strategy transforms temporal locality into spatial locality, exploited by the k^2-raster.

Both approaches address the same overarching goal—enhancing the raster times series compression—but from different perspectives: one aims to improve a well-established structure, and the other explores a new design paradigm. Presenting them together allows a unified discussion of their motivations, design principles, and experimental evaluations within a common framework of Compact Data Structures for raster time series.

This chapter is organized as follows. Section 3.2 explains the application of DP to the T-k^2-raster for optimal snapshot selection. Section 3.3 presents the ZT-k^2-raster structure. Section 3.4 describes the experimental framework, including configurations, implementation, and datasets. Section 3.5 reports and analyzes the experimental results. Finally, Section 3.6 summarizes the main conclusions.

3.2 Improving compression on the T-k^2-raster

Section 2.3.1 presents an overview of the functioning and representation of a temporal raster using the Heuristic T-k^2-raster. The approach classifies rasters as either snapshots or logs. As the heuristic reviews each raster in temporal order, it can only select the current or the previous raster as a new snapshot. This section describes a dynamic programming (DP) algorithm to select the optimal subset of rasters as snapshots that minimizes the space usage of the T-k^2-raster for the temporal raster. For each raster in the input sequence, this approach decides if a raster is a log or a snapshot exploring all previously defined snapshots and not only the previous or last snapshots, as used by the T-k^2-raster.

Let $\mathcal{M}$ be a raster time series of τ raster time instants, in which each raster is of size $n \times n$. The aim is to determine a snapshot subset $\mathcal{M}_s$ of $\mathcal{M}$ that represents the entire raster time series using a T-k^2-raster with minimal storage space required. Here, S represents the sorted index set of the selected subset ($\forall i \in S, i \in [1, \tau]$). The selection of this subset $\mathcal{M}_s$ is crucial for capturing the essential information of the raster time series while optimizing storage efficiency. For this, it is necessary to define the following operations:

- $ref(i)$: Computes the space required to represent the raster M_i using a k^2-raster.
- $c(i, j)$: Computes the space required to represent the raster M_i as a k^2-raster$_{log}$ using M_j as a reference.

The proposed approach identifies the subset $\mathcal{M}_s$ by solving an optimization problem using DP. Specifically, the goal is to find the index subset S that minimizes the following function:

$$\sum_{j=1}^{j \leq |S|} \left[ref(S[j]) + \sum_{i=S[j]+1}^{i<S[j+1]} c(i, S[j]) \right] \tag{3.1}$$

where $S[|S| + 1] - 1 = \tau$ for convenience.

Figure 3.1 illustrates an example of the application of DP over a particular raster time series. The gray cells in the list of rasters represent snapshots, while the white

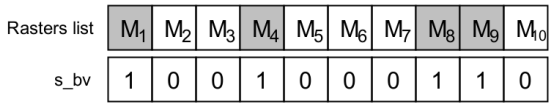


Figure 3.1: A raster time series example with their respective snapshots and logs rasters. The gray cells in the list of rasters represent snapshots, while the white cells represent logs. `s_bv` marks the rasters selected as snapshots (equivalent to `eqB`).

cells represent logs. In the implementation, the structure indicates the raster snapshots with a bitmap s_bv, i.e. $s\_bv[i]$ is 1 if raster i is a snapshot. In the example, the first raster time instant, M_1, is selected as a snapshot by default. Rasters M_2 and M_3 are encoded using M_1 as a reference, enabling the representation of the first three raster time instants using minimal space. As the difference between M_4 and M_1 is considerable, DP selects M_4 as a new snapshot. Besides, the difference between rasters M_4 and M_8 is considerable, enabling M_8 to be selected as a new snapshot. Hence, the selected subset of rasters is $\mathcal{M}_s = \langle M_1, M_4, M_8, M_9 \rangle$, with the corresponding sorted index set $S = \{1, 4, 8, 9\}$, and $|S| = 4$.

Let $\mathcal{M}[i, \ldots, j]$ denote a subproblem considering the raster intervals from M_i to M_j. We have two alternatives for selecting a snapshot. Firstly, we can consider M_i as the only snapshot where the space representation of the subproblem is minimal, which serves as the base case. The second option is to identify a position $r \in [i + 1, j]$ such that M_r represents the subproblem using the minimum space. In the case that DP chooses the second option and selects the snapshot M_r, the problem can be subdivided into two subproblems: $\mathcal{M}[i, \ldots, r-1]$ and $\mathcal{M}[r, \ldots, j]$, which can be solved recursively.

Equation 3.2 defines the optimization problem that needs to be solved for the subproblem.

$$\min\left(\left(ref(i) + \sum_{k=i+1}^{j} c(k,i)\right), \min_{i<r\leq j}\left[ref(i) + \sum_{k=i+1}^{r-1} c(k,i) + ref(r) + \sum_{k=r+1}^{j} c(k,r)\right]\right) \quad (3.2)$$

Consider a matrix $\mathbb{M}$ with dimensions of $\tau \times \tau$, where $\mathbb{M}[i, j]$ represents the minimum space required to represent the subproblem $\mathcal{M}[1, \ldots, i]$ using M_j as the last snapshot, with $j \leq i$. The matrix $\mathbb{M}$ can be computed using the recursive equation 3.3, with the calculation of each cell performed in row-major order from row 1 to τ, and from cell $[i, 1]$ to $[i, i]$ inside each row i.

$$\mathbb{M}[i,j] = \begin{cases} \min\limits_{1 \leq k \leq i-1} \mathbb{M}[i-1,k] + ref(i) & \text{if } i = j, \\ \min\limits_{1 \leq k \leq j} \mathbb{M}[j-1,k] + ref(j) + \sum\limits_{k=j+1}^{k \leq i} c(k,j) & \\ & \text{if } j < i \end{cases} \tag{3.3}$$

In the first part of Equation 3.3, the last raster is assumed to be a snapshot. To achieve this, the minimum space required to represent the raster time instants $\mathcal{M}[1, \ldots, i-1]$, along with the space required to represent M_i as a snapshot, is calculated. The second part of the equation assumes that M_j is the last snapshot. To compute the minimum space required for this case, the space needed to represent the raster time instants $\mathcal{M}[1, \ldots, j-1]$ is added to the space required to represent M_j as a snapshot, plus the space required to represent the raster time instants $\mathcal{M}[j+1, \ldots, i]$ encoded using M_j as a reference.

To compute the set of indexes S corresponding to the selected snapshots, we need to iterate through each row of the matrix $\mathbb{M}$ in reverse order, starting from the last row τ. For each row i, we compute the set of indexes $\{s_i | \min_{1 \leq s_i \leq i} \mathbb{M}[i, s_i]\}$, which correspond to the snapshots that minimize the space required to represent the subproblem $\mathcal{M}[1, \ldots, i]$. We collect all the different indexes s_i for each row i and store them in the set S. Finally, the selected snapshots are the set of rasters $\{M_s | s \in S\}$. The minimum space required to represent the entire $\mathcal{M}$ using the selected snapshots is given by $\min_{1 \leq j \leq \tau} \mathbb{M}[\tau, j]$.

The example in Figure 3.2 illustrates the process of selecting the last snapshot over the raster time series $\mathcal{M}$ to minimize its space representation. The black cells are not used because i is not less than j. The gray cells indicate the last snapshot selected for each row i that minimizes the space representation of $\mathcal{M}[1, \ldots, i]$. For instance, for $i \in [1, 3]$, the last snapshot selected is 1. However, for $i = 4$, the last snapshot selected

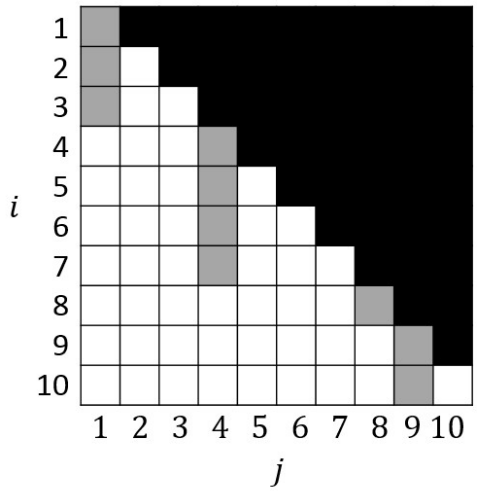


Figure 3.2: Matrix $\mathbb{M}$ example from Figure 3.1

is 4 as it minimizes the space required to represent $\mathcal{M}[1,\ldots,4]$.

To compute the entries of row $\mathbb{M}[6]$ of the matrix, we evaluate for each column $i \in [1,6]$, the minimum space required to represent $\mathcal{M}[1,\ldots,6]$ using M_i as the last snapshot. For example, to compute $\mathbb{M}[6,3]$, we first determine the minimum space required to represent $\mathcal{M}[1,2]$ using any snapshot k such that $1 \leq k \leq 2$ ($\min_{1\leq k\leq 2}\mathbb{M}[2,k]$). We then add the space required to represent M_3 as a snapshot ($ref(3)$) and the space required to represent $\mathcal{M}[4,\ldots,6]$ encoded by M_3 ($\sum_{k=4}^{k\leq 6} c(k,3)$).

3.3 A new proposal: ZT-k^2-raster

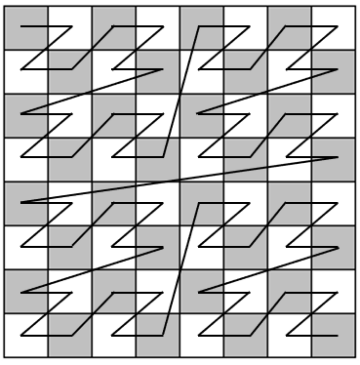


Figure 3.3: Example of Z-Order Curve on a raster of size 8×8

We introduce the ZT-k^2-raster, a Compact Data Structure (CDS) designed to represent raster time series efficiently. This structure employs a k^2-raster representing a two-dimensional version of the raster time series. Its application requires transforming the 3D raster time series into a 2D raster. To achieve this, a Space-Filling Curve (SFC) is applied to each raster, converting each 2D raster into a one-dimensional sequence. In our case, we apply the Morton curve, or Z-order [115], which traverses space recursively in a Z-shaped pattern (see Figure 3.3). The structure then arranges the linearized rasters sequentially, placing each one below the previous, forming the rows of the resulting 2D raster. The final image has dimensions $n \times r \times c$, where n is the number of rasters and $r \times c$ denotes the size of each raster in rows and columns.

During the construction of the k^2-raster, the raster to be compacted is virtually extended into a square raster with side x, where x is the smallest power of two that fully contains the raster. To determine x, the structure considers the larger value between the number of rows and columns. In general, $n < rc$, the structure selects the smallest integer d so that $rc \leq 2^d = x$. The virtual extension fills the raster with a single symbolic value. Typically, the temporal transformation of the raster produces an image with a much greater horizontal than vertical dimension, resulting in a considerable extension of the area. The k^2-raster structure leverages large regions of uniform values to enhance compression.

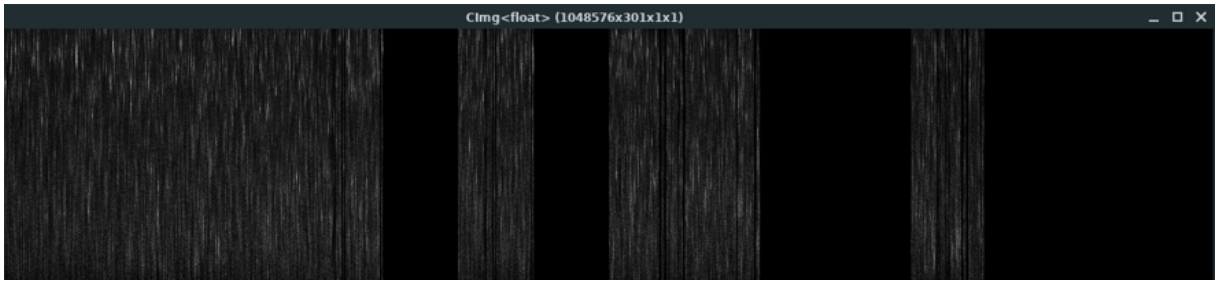


Figure 3.4: A raster time series linearized

Data locality is another property of the resulting image that the k^2-raster exploits. The Z order curve preserves the spatial locality of a raster, increasing the length of runs in the generated sequence. A run is a subsequence of equal values. Regions of similar values are produced by concatenating the sequences of contiguous rasters as consecutive rows of the generated image. This way, temporal locality is transformed

into spatial locality, which the k^2-raster leverages to improve compression. We can observe an example of the temporal raster linearization in Figure 3.4. The figure shows several vertical lines on a dark background. These lines represent high values in the raster time series, maintaining spatial locality. The dark background can represent low values or gaps due to the extent of each raster within the raster time series. The image shows a wide spatial locality that the ZT-k^2-raster can exploit.

3.4 Experimental framework

Server configuration: All the experiments were run on a dedicated Intel® Xeon® Gold 5320T CPU clocked at 2.30 GHz (40 physical cores) with cache sizes 1.9 MB (L1d), 1.3 MB (L1i), 50 MB (L2), and 60 MB (L3), and 252 GB of RAM. The operating system was Debian 11 with kernel 5.10.0-13-amd64. The C++ code was compiled with gcc version 10.2.1 and the -O3 optimizations. The Python code was executed with version 3.9.2.

Implementation code: We implemented both proposals using Fernando Silva's version 3.1 of the k^2-raster and T-k^2-raster[1]. Building on our DP proposal, we developed a new variant of the T-k^2-raster that employs the entropy-based heuristic k^2_H-raster as a base. Similarly, the implemented ZT-k^2-raster can work by applying the original k^2-raster or the entropy-based heuristic k^2_H-raster. The complete source code is available in a public repository[2].

Datasets: In this study, we use real world, synthetic, and semi-synthetic datasets. Regarding the real datasets, we used the NLDAS-2 collection. This collection was obtained from [166] and it was also used in the experimental process of [153]. This collection is a product of the North American Land Data Assimilation System (NLDAS). It includes information of precipitation and flows across North America from 1979 up to the present, such as surface temperature, humidity, and radiation, among other variables. These raster time series have an hourly time resolution and a spatial resolution

[1] `https://gitlab.lbd.org.es/fsilva/k2-raster`
[2] `https://gitlab.com/mmunocan/zt-k2raster`

of $1/8$ degrees. The specific dataset used in our experiments correspond to the time period from January to December 2018.

From the NLDAS-2 collection, we derived two sets of raster time series. The first, `APCP_X`, consists of the first X rasters extracted from the `APCP` dataset, with $X = 5, 10, 15, 50, 500, 1000, 1500$. The second, `PRES_X`, is obtained by dividing all values in the raster time series by 10^X, with $X \in [1..6]$. Dividing each value by a factor reduces its range, producing adjacent cells more similar and thereby increasing spatial locality. We selected this dataset for the `PRES_X` collection because it contains the largest values within the entire NLDAS-2 collection.

We also considered a second collection of raster time series, denoted `m_X`. This collection contains five raster time series representing a mouse liver tissue cells, where each raster time series (or channel represented with X) corresponds to a different protein from the same tissue [113]. We generated two different versions from each dataset: one with cell values divided by 10 and another with cell values divided by 100.

An additional variant, `m_X_Y`, was derived from the `m_X` collection. Here, X denotes the raster channel or ID, and Y indicates the filtering percentage applied to the temporal raster. Filtering involves removing $Y\%$ of the lowest values and replacing them with a single symbolic value. The filtering percentages used were $Y = 1, 2, 5, 10, 15, 20$, with $Y = 0$ representing the unmodified dataset. This variant was introduced because, in the context of this collection, only high values convey significant information, while an excessive number of low values adds noise. We used this collection for comparison with the ZT-k^2-raster.

Dataset	# Rows	# Cols	# Rasters	Minimum value	Maximum value	Unique values	% differences first raster	% differences last raster
APCP_5	224	464	5	-1	719	314	10.772	5.834
APCP_10	224	464	10	-1	719	345	10.996	4.337
APCP_15	224	464	15	-1	719	370	11.464	4.359
APCP_50	224	464	50	-1	1,369	479	12.730	4.000
APCP_100	224	464	100	-1	2,223	870	14.205	5.011
APCP_500	224	464	500	-1	3,967	1,873	18.973	9.146
APCP_1000	224	464	1,000	-1	4,293	2,487	20.006	10.009
APCP_1500	224	464	1,500	-1	4,405	2,709	20.482	10.635
APCP	224	464	2,665	-1	10,258	3,268	20.533	10.597
CAPE	224	464	2,664	-1	410,867	126,224	20.091	16.943
CONV	224	464	2,664	-1	100	102	2.948	1.850
DLWRF	224	464	2,664	9,919	47,911	24,167	77.350	40.711
DSWRF	224	464	2,664	-1	128,095	113,296	44.864	38.477
PEVAP	224	464	2,665	-79	185	263	68.746	19.557
PRES	224	464	2,664	6,301,609	10,502,674	2,057,044	77.392	77.386
SPFH	224	464	2,664	-1	2	4	3.596	0.262
TMP	224	464	2,665	23,421	31,223	6,890	77.356	76.647
UGRD	224	464	2,664	-2,163	2,147	3,815	77.281	76.205
VGRD	224	464	2,664	-2,004	2,182	3,824	77.286	76.202
PRES_1	224	464	2,664	630,160	1,050,267	401,505	77.390	77.343
PRES_2	224	464	2,664	63,016	105,026	41,212	77.363	76.882
PRES_3	224	464	2,664	6,301	10,502	4,178	77.098	72.189
PRES_4	224	464	2,664	630	1,050	421	74.457	35.931
PRES_5	224	464	2,664	63	105	43	53.137	3.894
PRES_6	224	464	2,664	6	10	5	12.189	0.490
m1_0	752	752	301	22	3,213	2,550	99.763	99.607
m1_1	752	752	301	3	4,095	3,200	98.564	97.831
m1_2	752	752	301	28	4,095	3,601	98.916	98.807
m1_3	752	752	301	0	4,095	4,095	97.761	96.437
m1_4	752	752	301	34	4,095	3,322	99.751	99.521
m1_0 div 10	752	752	301	2	321	286	97.636	96.065
m1_1 div 10	752	752	301	0	409	401	90.368	86.691
m1_2 div 10	752	752	301	2	409	408	89.288	88.240
m1_3 div 10	752	752	301	0	409	410	87.966	83.574
m1_4 div 10	752	752	301	3	409	378	97.520	95.233
m1_0 div 100	752	752	301	0	32	32	77.446	64.133
m1_1 div 100	752	752	301	0	40	41	32.689	19.634
m1_2 div 100	752	752	301	0	40	41	50.443	43.169
m1_3 div 100	752	752	301	0	40	41	43.012	33.865
m1_4 div 100	752	752	301	0	40	41	77.101	59.123

Table 3.1: Description of real world and semi-synthetic rasters collections

Table 3.1 provides a detailed description of the real and semi-synthetic datasets. The table also reports the percentage of variation of each raster relative to the first raster and its immediate predecessor, as an indicator of the raster time series variability. Higher variability implies lower temporal locality, which increases snapshot selection and challenges compression. The NLDAS-2 collection exhibits highly variable datasets, while the m_X collection shows even greater variation. Dividing the values, as in the PRES_X and m_X variants, reduces the percentage of variation.

In addition, we included a synthetic datasets collection of raster time series. These datasets were generated using the Zipf distribution [146], which assigns the frequency of occurrence of each value as $1/n^{\alpha}$, where n is the nth value and α determines the slope of the distribution curve. Larger values of α produce a narrower range of generated values. Two parameters are defined to construct a temporal raster: temporal distribution (α_t) and spatial distribution (α_e). The first raster of the raster time series requires $\alpha = \alpha_e$ to generate its values. Subsequent rasters generate their values by adding differences from the previous raster. The differences require $\alpha = \alpha_t$ according to the Zipf distribution. Thus, α_t and α_e control temporal and spatial locality, respectively.

Dataset	# Rows	# Cols	# Rasters	Minimum value	Maximum value	Unique values	% differences first raster	% differences last raster
α_t_2	512	512	2,048	1	472,877,674	652,557	99.924	39.204
α_t_5	512	512	2,048	1	180	173	98.679	3.562
α_t_10	512	512	2,048	1	13	13	57.330	0.099
α_e_2	512	512	2,048	1	432,120	1,164	57.304	0.099
α_e_5	512	512	2,048	1	22	21	57.320	0.099
α_e_10	512	512	2,048	1	13	13	57.330	0.099
r_256	256	256	2,048	1	12	12	57.226	0.099
r_512	512	512	2,048	1	13	13	57.330	0.099
tr_128	512	512	128	1	7	7	6.143	0.100
tr_256	512	512	256	1	8	7	11.691	0.099
tr_512	512	512	512	1	8	8	21.589	0.099
tr_1024	512	512	1,024	1	10	10	37.278	0.099
tr_2048	512	512	2,048	1	13	13	57.330	0.099

Table 3.2: Description of synthetic raster collection

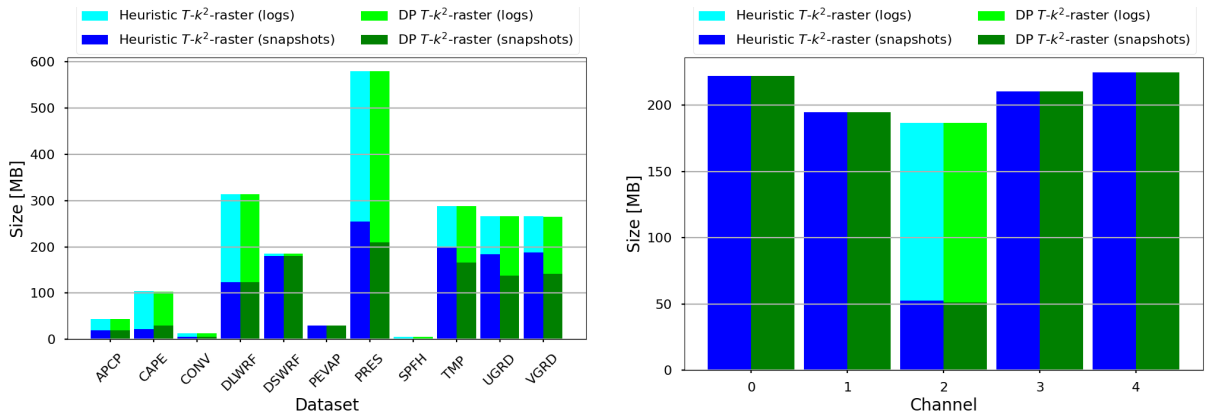
Table 3.2 summarizes the synthetic datasets. Four parameters were varied: α_t, α_e, raster size, and number of rasters. We vary one parameter for each dataset, while the others are assigned the highest value considered. For each dataset, we generate five raster time series. The values presented for each dataset correspond to the average of the five generated time series rasters.

3.5 Experimental results

This section details the experimental evaluation. Section 3.5.1 focuses on the results of the dynamic programming algorithm, whereas Section 3.5.2 examines the performance

of the ZT-k^2-raster.

3.5.1 Dynamic programming comparison



(a) NLDAS-2 (b) m_X

Figure 3.5: Temporal k^2-raster sizes (in [MB]) comparison between Heuristic and dynamic programming approaches on real-world collections. Channel value in m_X represents a different protein from the mouse liver tissue represented.

Figure 3.5 compares the Heuristic T-k^2-raster with its DP variant on real-world datasets. The DP version employs the k^2_H-raster as its base. The size of the compacted raster time series remains similar across both approaches. Likewise, snapshot memory is comparable between the two versions in most cases. Typically, the snapshot sizes in the heuristic approach are similar to or larger than those obtained with the DP approach. Only in CAPE for NLDAS-2 collection, DP produces larger snapshots than the heuristic. For semi-synthetic collections, we highlight PRES_X, where an increase in the division factor results in larger snapshot sizes in DP compared to the heuristic.

Table 3.3 compare the number snapshot selected between the Heuristic and DP strategies for NLDAS-2 collection. The memory usage is directly related to the number of selected snapshot rasters, which is generally similar across both approaches. The following datasets present fewer snapshots in the DP version than the heuristic version: PRES, TMP, UGRD, and VGRD. In those cases, the final structure size between the two methods remains almost unchanged. For SPFH, the DP version selects 159 snapshots while the heuristic version selects only 14 snapshots. However, the overall structure's

size remains similar.

Figure 3.6 compares the Heuristic T-k^2-raster with its DP variant on semi-synthetic and synthetic dataset collection respectively. We can observe a particular behavior: DP tends to select more snapshots when the alphabet size decreases, but this strategy does not improve the raster compression. In particular, we can observe this phenomenon in PRES_X (see Figure 3.6b), m_X divided (see Figure 3.6c) and synthetic Collections (see Figure 3.6d).

In Figure 3.6d, we can observe a significant difference in the final size of the structures. The DP version achieves compression improvements of up to 55% compared to the heuristic. As illustrated in the fourth row, this advantage is maximized when high α_t and low α_e. These results indicate that DP performs exceptionally well when temporal locality is high and spatial locality is low. Moreover, the difference between the number of selected snapshots and the memory used is larger for DP.

Dataset	Heuristic T-k^2-raster	DP T-k^2-raster	Common snapshots
APCP	888	889	885
CAPE	471	673	201
CONV	633	723	624
DLWRF	888	888	888
DSWRF	1,544	1,514	1,509
PEVAP	889	889	889
PRES	1,162	955	634
SPFH	14	159	3
TMP	1,820	1,527	1,312
UGRD	1,839	1,372	1,106
VGRD	1,879	1,416	1,162

Table 3.3: Number of snapshots generated comparing Heuristic and Dynamic Programming

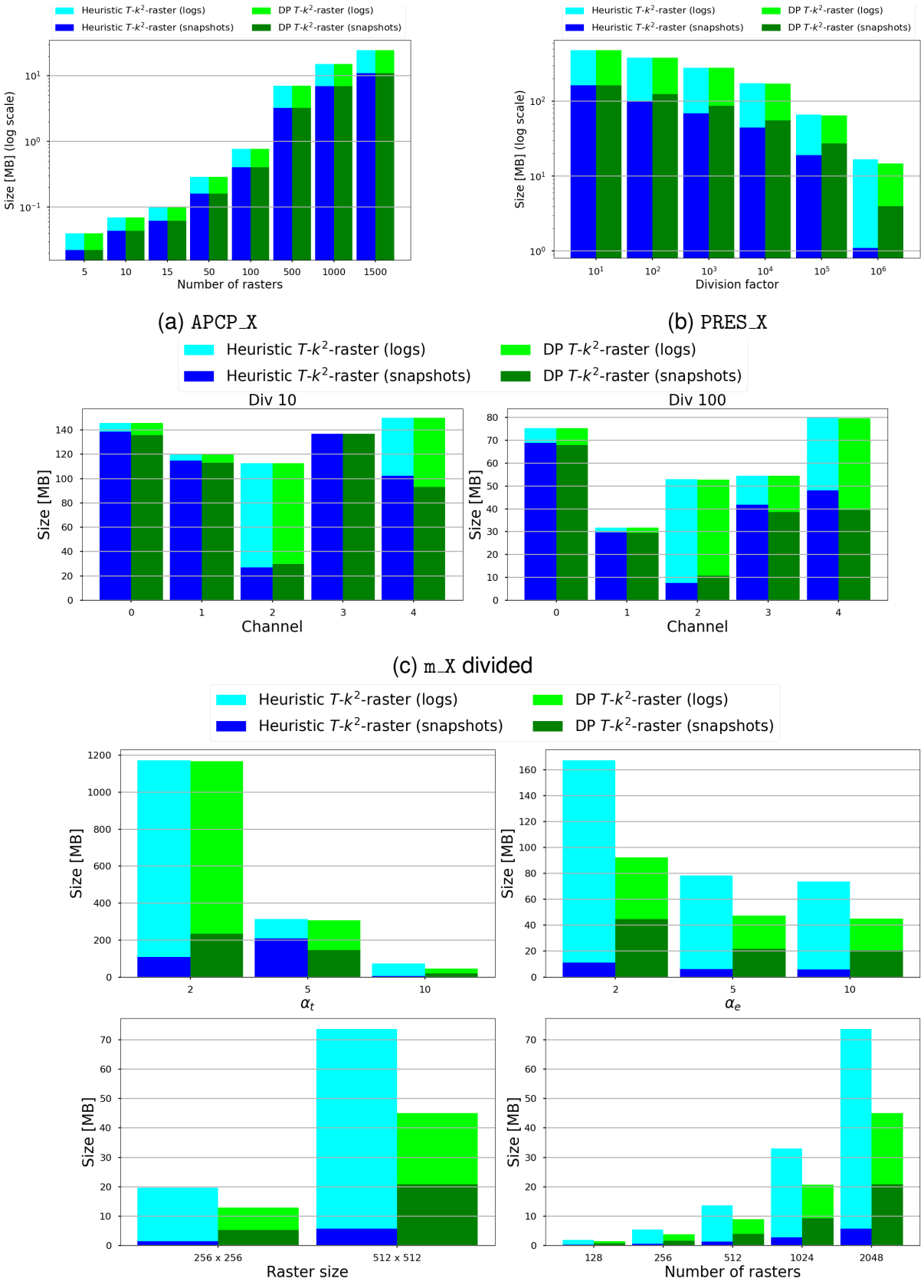
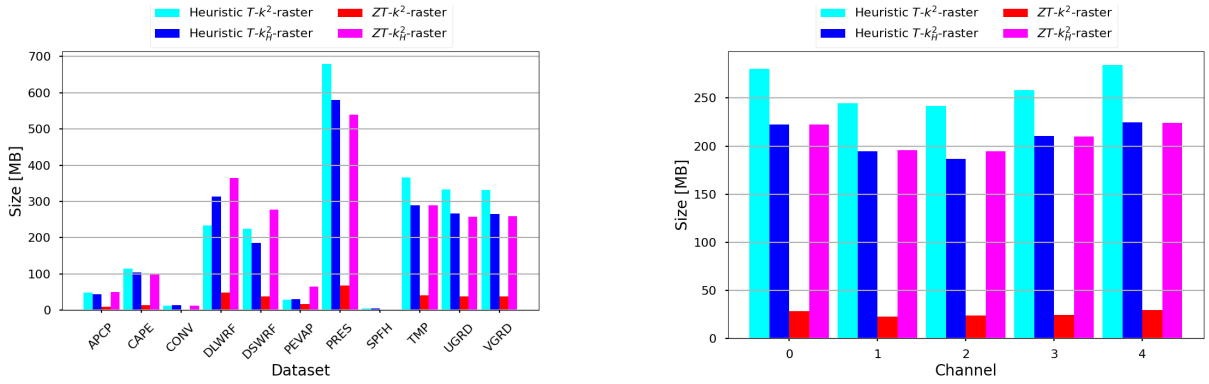


Figure 3.6: Temporal k^2-raster sizes (in [MB]) comparison between Heuristic and dynamic programming approaches on semi-synthetic and synthetic collections. Channel value in m_X represents a different protein from the mouse liver tissue represented. The synthetic collection using Zipf distribution, varying spatial distribution (α_e), temporal distribution (α_t), raster size and number of rasters.

In conclusion, after comparing the T-k^2-raster heuristic with the DP approach, we observed that the approach did not enhance performance on real-world datasets. Heuristic approaches are just as good as the DP approach, especially with real-world datasets. On the other hand, the DP method reduced the size of the temporal raster in specific synthetic datasets. This behavior occurs when the temporal raster presents high temporal locality and low spatial locality.

3.5.2 ZT-k^2-raster comparison



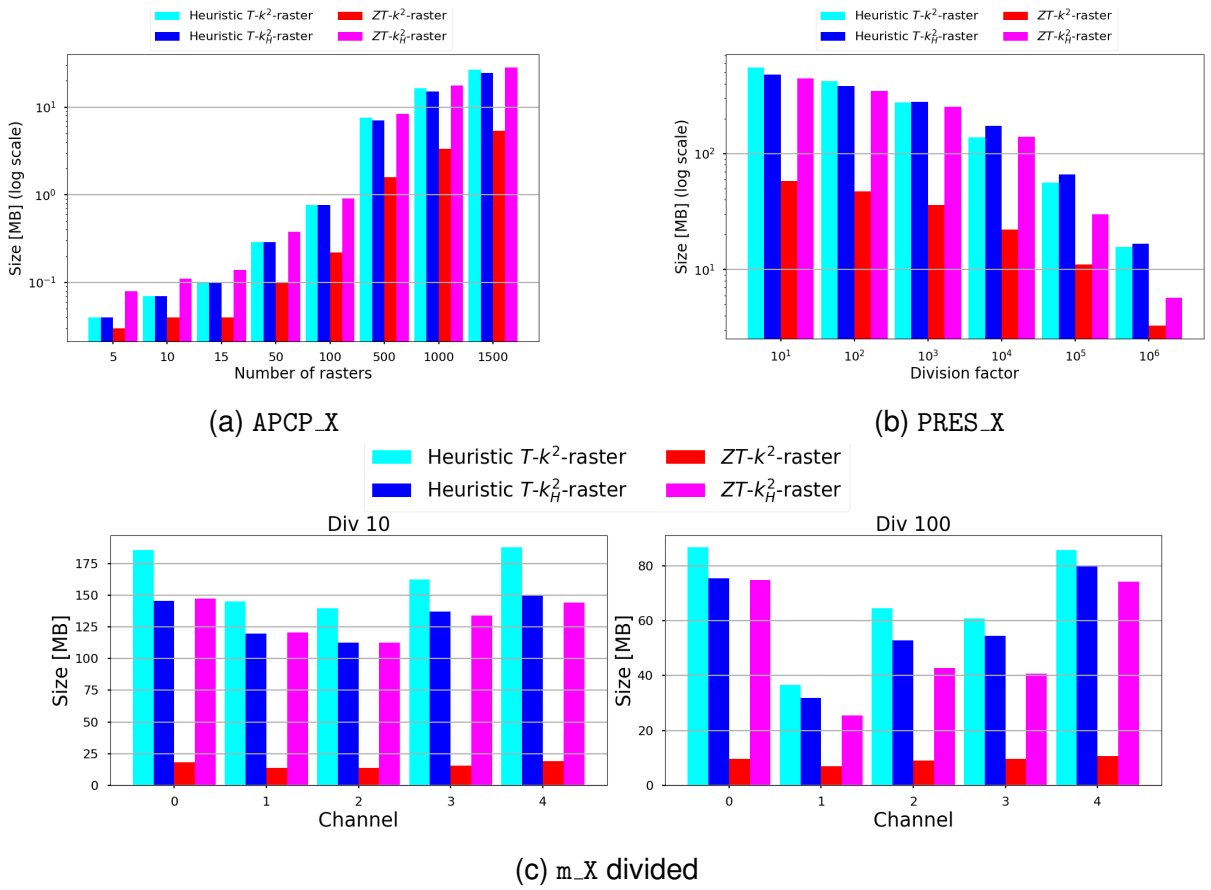
(a) NLDAS-2 (b) `m_X`

Figure 3.7: Structures sizes (in [MB]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on real-world collections. Channel value in `m_X` represents a different protein from the mouse liver tissue represented.

Figures 3.7, 3.8, 3.9, and 3.10 present the resulting sizes after constructing the T-k^2-raster and the ZT-k^2-raster across all dataset collections introduced in Section 3.4. We denote the CDS variant that contains the entropy-based dictionary k^2-raster as T-k^2_H-raster and ZT-k^2_H-raster.

The results show that the ZT-k^2-raster is smaller than the T-k^2-raster in its standard and entropy-based versions in most cases, with the standard version generating the most significant differences. In Figure 3.7b, the ZT-k^2-raster reduces structure size by up to 90% compared to the T-k^2-raster in the standard version. However, when using the entropy-based dictionary versions, the ZT-k^2_H-raster is similar, and in some cases even greater, compared to the T-k^2_H-raster. This situation also occurs, for example, in Figure 3.7a, for datasets such as `APCP`, `DLWRF`, `DSWRF`, and `PEVAP`.

These findings indicate that the ZT-k^2-raster is a competitive structure for raster time series compression compared to the T-k^2-raster. The version without the entropy-based dictionary performs remarkably well, since the dictionary in the ZT-k^2_H-raster requires more space than in the T-k^2_H-raster. This behavior is because the embedded k^2_H-raster within the ZT-k^2_H-raster represents a larger raster compared to the diverse rasters that represent the T-k^2-raster, producing a larger structure with more levels, increasing the dictionary size.



(a) APCP_X (b) PRES_X

(c) m_X divided

Figure 3.8: Structures sizes (in [MB]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on semi-synthetic collections. Channel value in m_X represents a different protein from the mouse liver tissue represented.

As shown in Figure 3.10, applying filtering (removing $Y\%$ of the lowest values and replacing them with a single value), reduces the size of all structures, with filtering percentages above 10% yielding significant compression improvements.

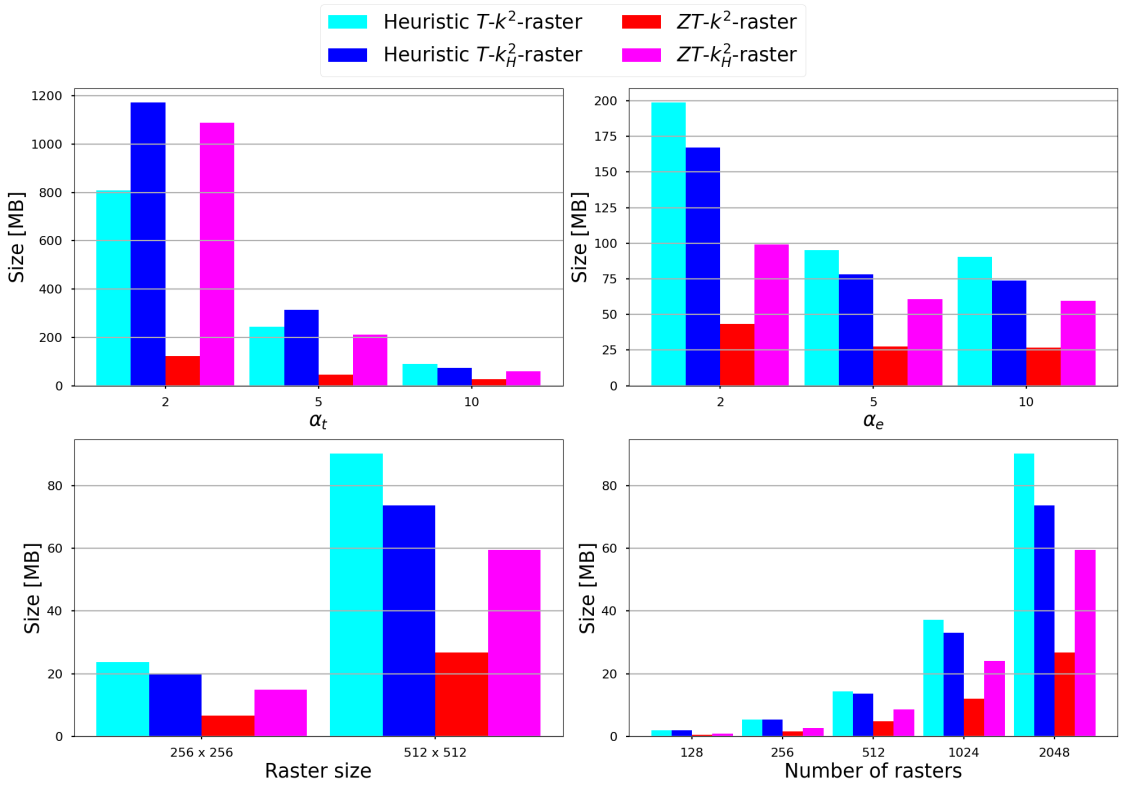


Figure 3.9: Structures sizes (in [MB]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on synthetic collection using Zipf distribution, varying spatial distribution (α_e), temporal distribution (α_t), raster size and number of rasters.

Figures 3.11, 3.12, 3.13, and 3.14 report construction times. The ZT-k^2-raster requires substantially longer construction times than the T-k^2-raster. On average, building the T-k^2-raster takes around 100 seconds, whereas constructing the ZT-k^2-raster requires approximately 100,000 seconds. For a brief theoretical analysis, the T-k^2-raster requires $O(c \times \tau)$ for a raster with τ raster timestamps and $c = n \times m$ cells per raster with n rows and m cols. In contrast, ZT-k^2-raster requires $O((\max(c, \tau))^2)$ for the subjacent raster of size $\max(c, \tau) \times \max(c, \tau)$.

Figures 3.15, 3.16, 3.17, and 3.18 show the performance of the *getCell* query. While the gap between both structures is smaller than in construction time, queries on the ZT-k^2-raster still take longer than on the T-k^2-raster. For a brief theoretical analysis, querying in a T-k^2-raster requires select the current raster timestamp in constant time and selecting the raster cell inside in $O(\log \max(m, n))$ time. On the other hand, querying in a ZT-k^2-raster requires selecting the raster cell in the subjacent raster in

$O(\log \max(n \times m, \tau))$ time.

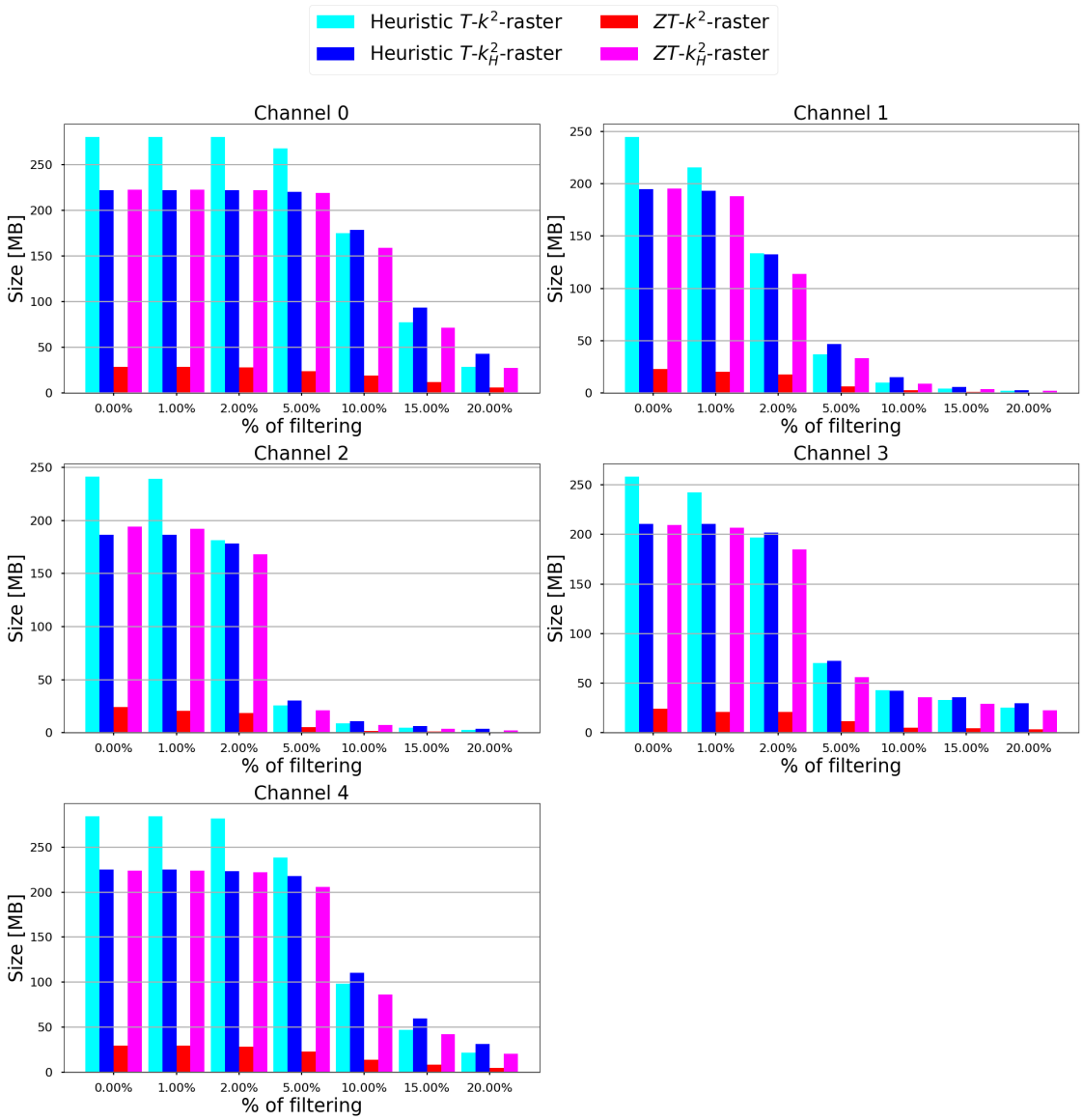
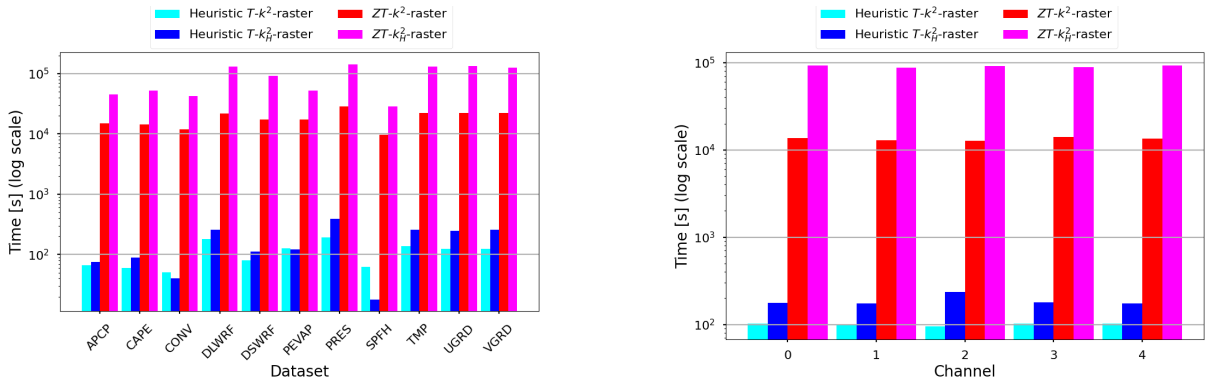
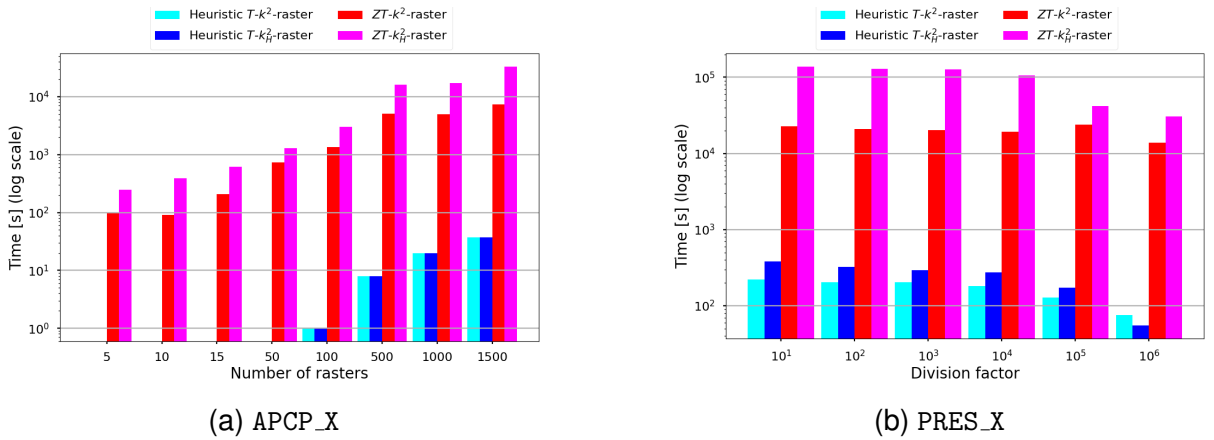


Figure 3.10: Structures sizes (in [MB]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on `m1` filtered collections. Channel value in `m_X` represents a different protein from the mouse liver tissue represented.

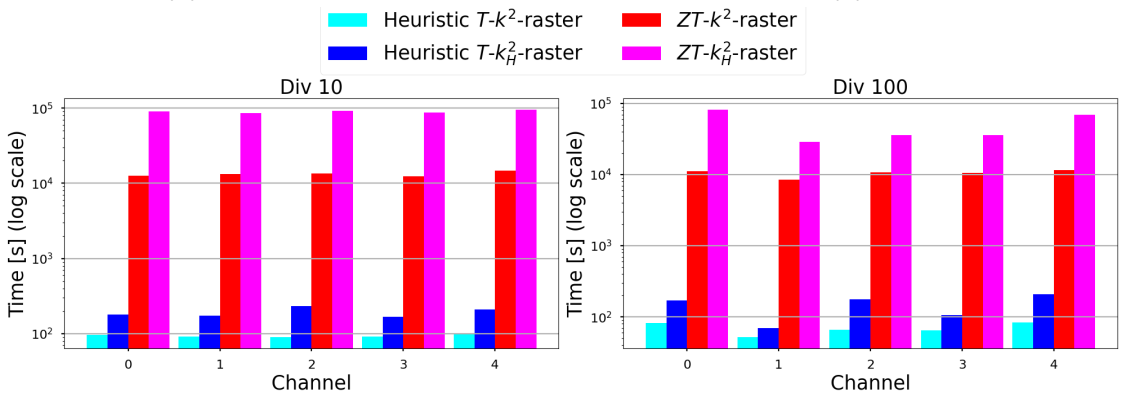


(a) NLDAS-2 (b) m_X

Figure 3.11: Construction time (in [s]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on real-world collections. Channel value in m_X represents a different protein from the mouse liver tissue represented.



(a) APCP_X (b) PRES_X



(c) m_X divided

Figure 3.12: Construction time (in [s]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on semi-synthetic collections. Channel value in m_X represents a different protein from the mouse liver tissue represented.

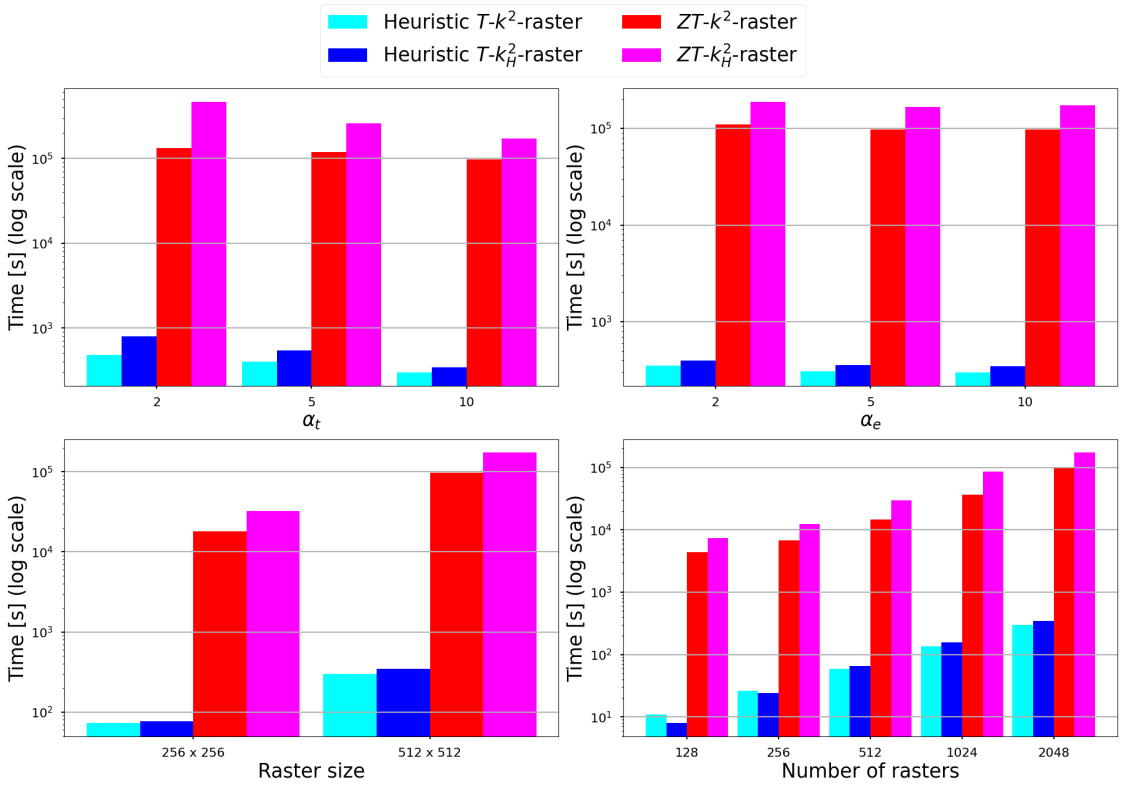


Figure 3.13: Construction time (in [s]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on synthetic collection using Zipf distribution, varying spatial distribution (α_e), temporal distribution (α_t), raster size and number of rasters.

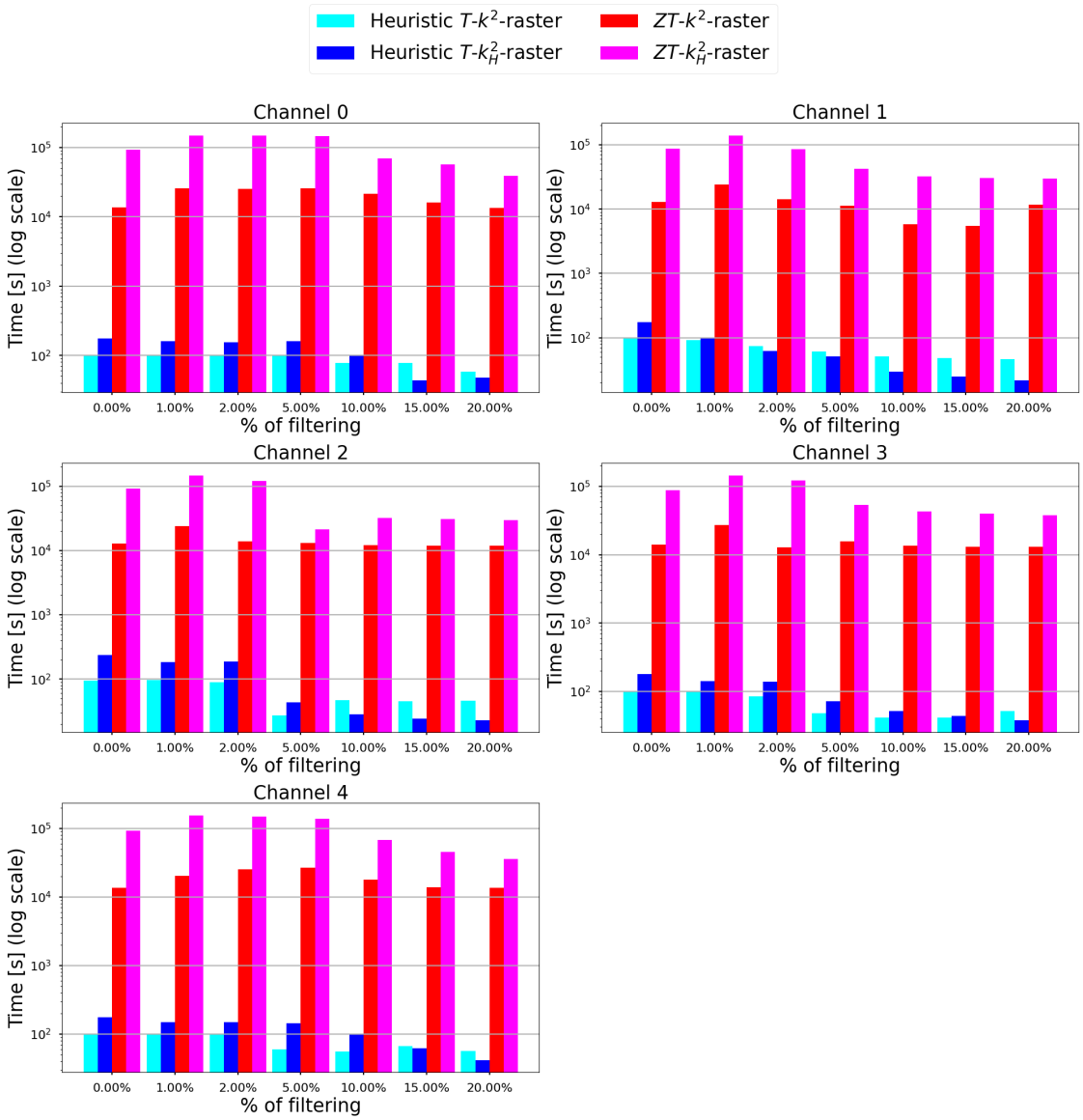
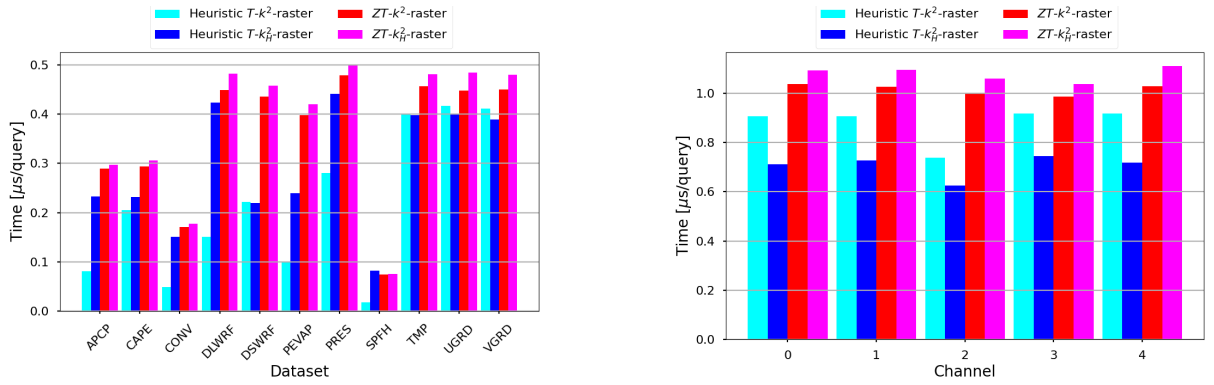
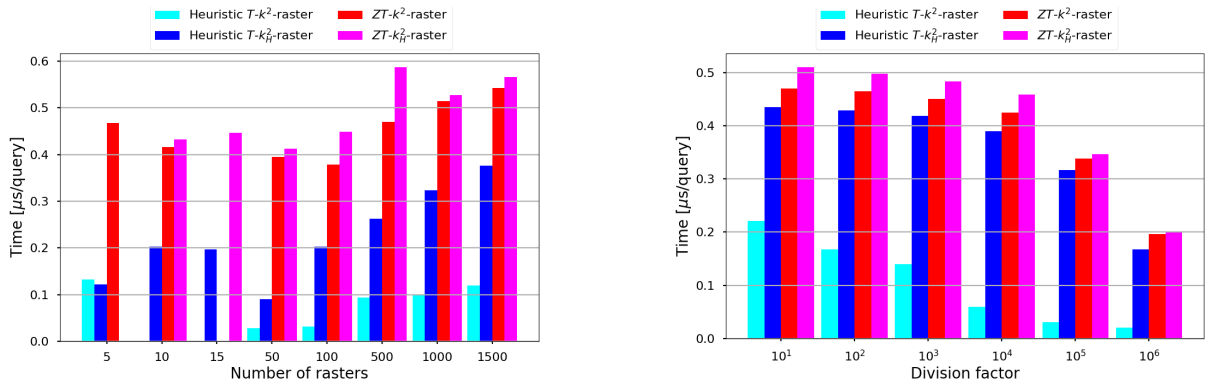


Figure 3.14: Construction time (in [s]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on `m1` filtered collections. Channel value in `m_X` represents a different protein from the mouse liver tissue represented.

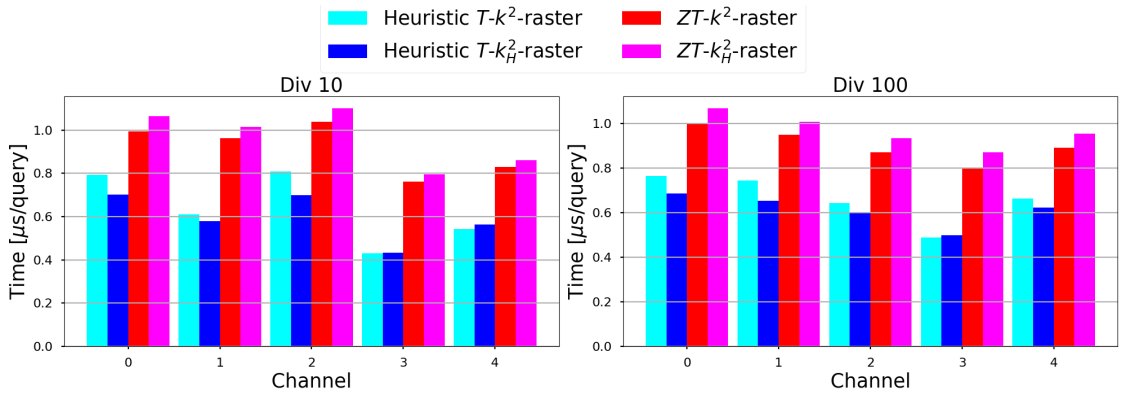


(a) NLDAS-2 (b) m_X

Figure 3.15: *getCell* time (in [μs/query]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on real-world collections. Channel value in m_X represents a different protein from the mouse liver tissue represented.



(a) APCP_X (b) PRES_X



(c) m_X divided

Figure 3.16: *getCell* time (in [μs/query]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on semi-synthetic collections. Channel value in m_X represents a different protein from the mouse liver tissue represented.

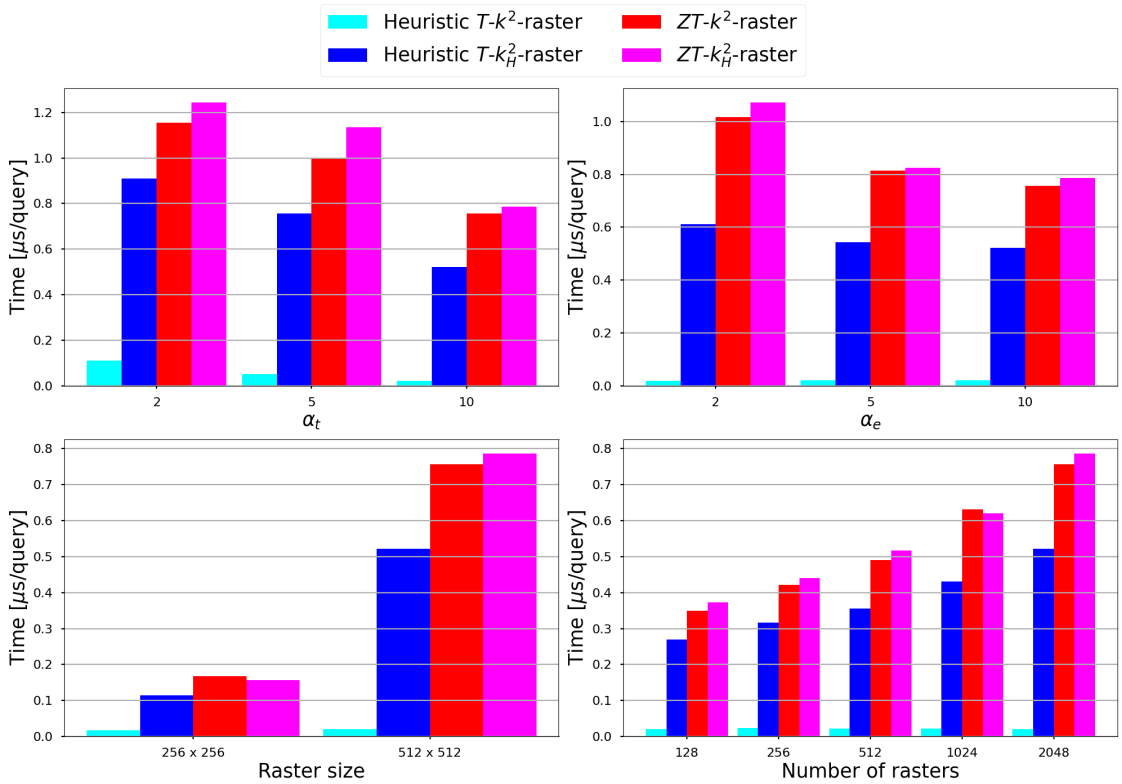


Figure 3.17: *getCell* time (in [μs/query]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on synthetic collection using Zipf distribution, varying spatial distribution (α_e), temporal distribution (α_t), raster size and number of rasters.

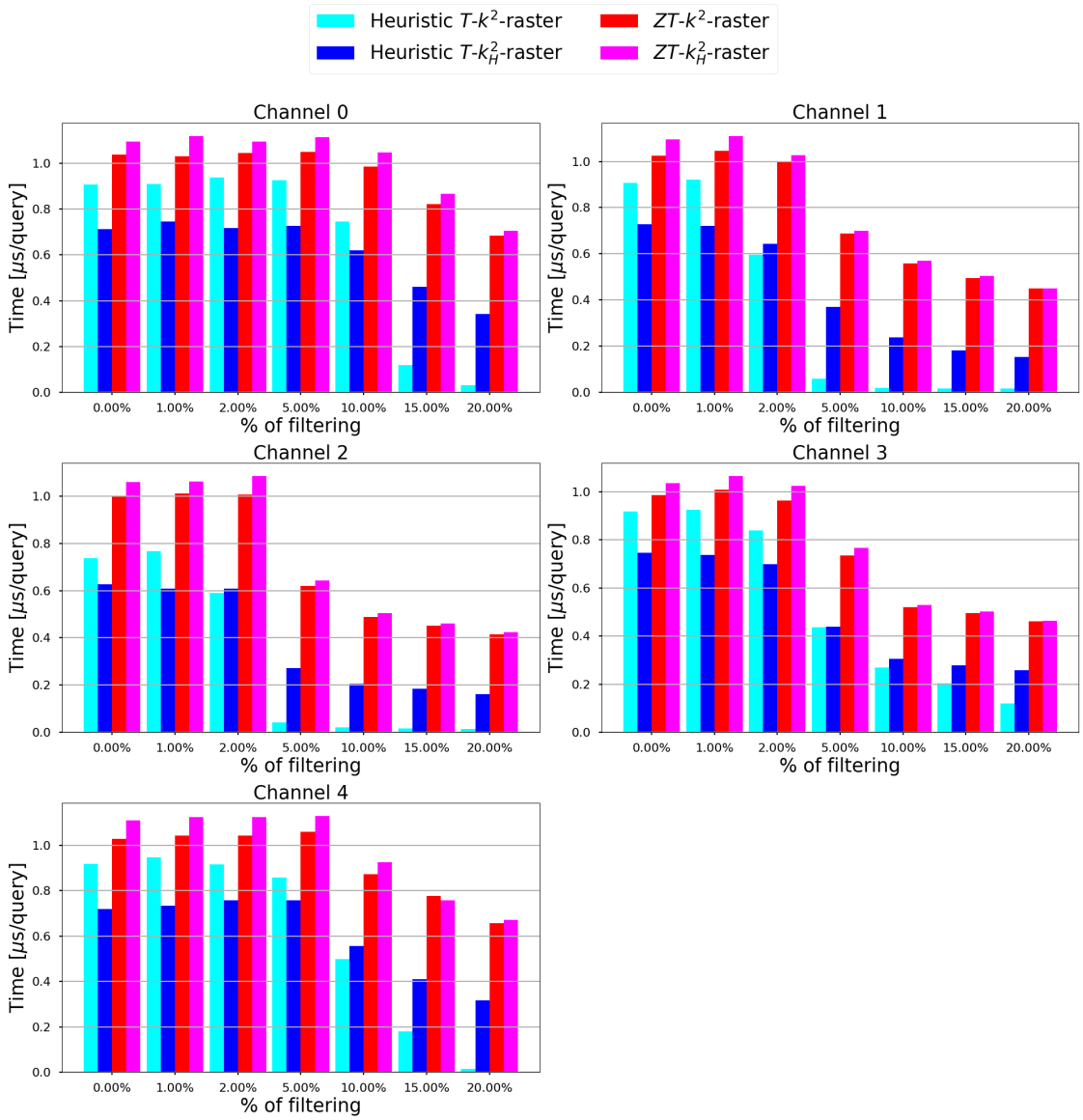


Figure 3.18: *getCell* time (in [μs/query]) comparing Heuristic T-k^2-raster and ZT-k^2-raster on `m1` filtered collections. Channel value in `m_X` represents a different protein from the mouse liver tissue represented.

In summary, the ZT-k^2-raster is highly competitive in terms of compression, but its construction and query times are considerably higher. This is explained by its design: the ZT-k^2-raster builds a single large k^2-raster with many levels, whereas the T-k^2-raster constructs multiple smaller k^2-rasters with fewer levels. As a result, the larger

raster represented by the ZT-k^2-raster improves compression but increases construction and query times.

3.6 Conclusions

This chapter proposes two approaches enhancing raster time series compression. First, we introduced a dynamic programming (DP)-based version to compare against the heuristic, evaluating its optimality and potential improvements. Subsequently, we presented the ZT-k^2-raster, a CDS that linearizes each raster, joins each sequence as a row in a new raster, and compacts the result with a k^2-raster. The ZT-k^2-raster achieves compression advantages due to the large virtual extent of the raster during k^2-raster construction and the transformation of spatial and temporal locality into spatial locality in the resulting raster.

The comparison between the heuristic and DP revealed no significant differences in compression size across real-world and semi-synthetic dataset collections. Minor differences appear when reducing the alphabet size, where DP tends to select more raster snapshots than the heuristic, which increases snapshot sizes but leaves the final structure sizes similar to the heuristic. In contrast, a substantial gap emerges in synthetic datasets: DP achieves better compression under high temporal locality than the heuristic.

When comparing T-k^2-raster and ZT-k^2-raster, results show that our proposal is competitive regarding compression, but not in construction or query times. The ZT-k^2-raster improves compression, notably when the underlying k^2-raster does not employ the entropy-based dictionary.

We also evaluated a filtering technique applied to the `m1` collection, which discards the smallest values. Results indicate that even a low filtering percentage (below 10%) can improve compression effectively. This approach is especially beneficial in scenarios where higher values carry the relevant information, while lower values primarily represent noise.

In summary, the heuristic achieves compression close to the optimal solution provided by DP, though synthetic experiments highlight situations where DP offers a clear

advantage. Meanwhile, the ZT-k^2-raster demonstrates strong compression capabilities but suffers from prohibitively high construction costs due to the deeper k^2-raster levels it generates. Finally, minimum-value filtering is a reasonable complementary strategy in contexts where high values represent information and low values represent noise.

Chapter 4

Clustering-based compression for raster time series

4.1 Introduction

The Heuristic T-k^2-raster is a Compact Data Structure (CDS) representing raster time series [153, 33]. In this structure, each raster is classified as either a **snapshot** or a **log** and represented using a variant of the k^2-raster. This structure includes a heuristic for efficiently and automatically selecting snapshots and logs to improve data compression. However, this heuristic has a constraint regarding the snapshot selection. For a raster log, only the latest previously selected snapshot in chronological order can be its respective snapshot reference. This limitation impedes the selection of the best snapshot candidate to improve the data representation. This is important in domains where the variable under study exhibits cyclic behavior, meaning that the values represented at one point in time can repeat or be very similar at other points in time (e.g., the temperature at a specific hour of the day may be similar on other days during the same season). In other words, applying clustering to the T-k^2-raster allows us to exploit both spatial and temporal locality and the cyclic property.

This chapter comprehensively studies the heuristic used for selecting snapshots in the Heuristic T-k^2-raster. To further reduce the space usage of the data structure, we explore alternative representations that eliminate the snapshot selection constraint. Specifically, we explore the application of clustering algorithms using a distance measure based on Hamming distance. Using clustering enables us to choose, as snapshots, those rasters that are the centroids of the clusters, and therefore, the representation is not restricted to following the time-ordered rasters. However, to enable the same query support of the T-k^2-raster, an additional integer vector is needed to identify, for each raster, its respective snapshot or cluster centroid. Our results show that clustering can improve the compression performance of the Heuristic T-k^2-raster when the raster time series shows cyclic behavior, keeping the query support performance.

This chapter is structured as follows. Section 4.2 provides essential background information, introducing key concepts necessary for understanding the study. In Section 4.3, relevant related work is discussed. Section 4.4 elaborates on the application of clustering to the T-k^2-raster. The experimental process, results, and corresponding discussion are presented in Section 4.5. Finally, Section 4.6 presents the conclusions drawn from this study and outlines potential future directions.

4.2 Background

This section provides an overview of the specific background required to understand our study. The principal subject is clustering techniques, revised in Section 4.2.1.

4.2.1 Clustering

Clustering is a technique that aims to group elements, referred to as "points", into clusters based on a distance metric. The goal is to create groups where each group contains close points, and the distance between points in different groups is high. Clustering is fundamental in many fields, such as machine learning, data mining, and pattern recognition enabling the discovery of hidden patterns, performing exploratory data analysis, and reducing the dimensionality of large datasets [11]. Some well-known distance metrics include Euclidean, Hamming, Edit, and Jaccard distances, where choosing an appropriate distance metric is crucial to obtain meaningful and effective clustering results [139].

Two popular clustering strategies are hierarchical and partition-based clustering. Hierarchical clustering, or agglomerative clustering [139], starts by defining each data point as a separate cluster. Next, it proceeds to iteratively merge the two closest clusters into a single larger cluster until the desired number of clusters is reached. The hierarchical clustering methods can be classified based on how they compute the distance metric. Some commonly used methods include Single-link clustering, Complete-link clustering, and Average-link clustering [139]. In this study, we apply the Complete-link scheme, which measures the similarity between two clusters by considering the maximum distance between any two points, one from each cluster.

On the other hand, the partition-based clustering scheme employed in this study is the k-means algorithm [134, 158]. The algorithm starts by selecting k representative points as the initial centroids of the k clusters. Subsequently, the algorithm assigns each non-representative point to the closest cluster based on the distance between the point and the cluster centroid. Since including new points affects the cluster centroid, the algorithm performs multiple iterations over the entire set of points until they remain in the assigned cluster. The number of iterations can be set to a fixed number or performed until the algorithm converges. The selection of the initial k centroids is a crucial aspect of the k-means algorithm, as it can significantly impact its effectiveness. Typically, the initial k points are selected randomly, but this approach can be problematic if two points too close to each other are selected. The k-means++ is a variant of the k-means algorithm to reduce the effect of randomness in selecting the k points that represent the clusters [14]. The method biases the selection by randomly choosing the first representative point from the data points and then selecting the following points with probability proportional to the square of their distance from the nearest already chosen center. This approach helps to improve the selection of the first k points by choosing points farther apart. Despite the additional initialization cost, the k-means++ algorithm converges faster and produces better results than the standard k-means algorithm, reason why we use it in our study.

In addition, the effectiveness of many clustering techniques also depends on selecting the number of clusters. Several indices have been proposed to estimate the number of clusters, including the Silhouette index, which compares the intracluster and intercluster distances of the partitioning and it is defined as follows [141]: Given a point p_i, compute the intracluster distance $a(i) = \frac{1}{|C_I|-1}\Sigma_{j\in C_I, i\neq j}d(i,j)$ and the intercluster distance $b(i) = \min_{J\neq I}\{\frac{1}{|C_J|}\Sigma_{j\in C_J}d(i,j)\}$, where C_I is the cluster that contains p_i, C_J is another cluster and $d(i,j)$ is the distance applied to points p_i and p_j. Finally, the Silhouette value of point p_i, named $s(i)$, can be computed by applying Equation 4.1:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{4.1}$$

The Silhouette value assesses the classification quality of an individual data point within its cluster, while the Silhouette index S represents the average of these values

across all points in a clustering solution and estimates the clustering quality. The Silhouette index ranges between -1 and 1, where a higher value indicates that the points are well-clustered. Hence, the ideal number of clusters k is such that maximizes the average Silhouette index.

4.3 Related Work

This section provides an overview about related applications of clustering techniques to raster data.

4.3.1 Applications of clustering to raster data

There are several works related to the application of clustering on raster data. Alkathiri *et al*. [6, 7] investigated the utilization of k-means clustering for processing multi-spectral geo-spatial raster data in a Hadoop environment. Alzaghoul *et al*. [10] applied clustering to rasters representing Digital Elevation Models, aiming to identify hidden patterns, uncover relationships, and discover clusters of elevation values. Image compression of RGB photos was addressed by authors in [163], where clustering algorithms were employed. Kiran [87] applied clustering to discover knowledge from raster data. Mariani *et al*. [109] presented a distributed clustering algorithm to handle big data rasters in a decentralized manner. Aghaee *et al*. [2] applied clustering to predict geological lineaments using topographic, magnetic, and gravity raster data. Wu *et al*. [165] proposed a pixel clustering-based method to enhance the efficiency of mining spatial sequential patterns from raster serial remote sensing images (SRSI). These studies demonstrate the wide range of applications and the potential benefits of employing clustering techniques in the analysis and processing of raster data.

Sisodiya *et al*. [155] applied clustering on raster data compacted in a k^2-raster [97, 98]. The authors aimed to overcome memory limitations associated with traditional clustering methods when dealing with large datasets containing raster values. The findings of this study highlight the potential of employing the k^2-raster and clustering methods to analyze raster data in a more efficient and scalable manner. Their research demonstrated that the proposed approach, based on a CDS, effectively addressed the

challenges of data representation and scalability in clustering.

4.4 Using clustering to improve the T-k^2-raster

In Section 4.2, we discussed different approaches for selecting snapshots to succinctly represent a raster time series using a T-k^2-raster [153, 33]. The principal characteristic of such techniques is that a raster log uses as a reference a preceding raster snapshot in time-order. However, this limitation restricts exploring alternative combinations, such as referencing a subsequent snapshot that may be more similar than any of the previous ones. By considering these alternative strategies, it becomes possible to discover new combinations of snapshots and logs that can effectively minimize the size of the data structure.

Depending on the temporal locality, the distance between neighboring or nearby rasters is expected to be less than between distant rasters. However, in certain cases, this pattern may not hold or additional patterns may also exist. For instance, if the raster time series reflects a cyclical variable, the values may repeat every certain number of rasters. Therefore, it is crucial to identify such patterns and group similar rasters together, regardless of the number of rasters that separate them.

To address this, we introduce the application of the clustering technique to enable the selection of snapshots that correspond to each log. This process aims to reduce the size of the T-k^2-raster used to represent the data.

For the application of clustering analysis to our raster time series problem, individual rasters are considered as “points”. The order of the rasters within the time series is disregarded, allowing them to be rearranged and grouped according to their similarity. This approach enables us to focus on the similarities among the rasters and disregard their temporal dependence.

While storing the referenced snapshot for each log is necessary to achieve a complete representation of a T-k^2-raster, the additional space required for this purpose is insignificant. Each raster in the temporal raster requires a constant value to indicate its corresponding snapshot. In Figure 4.1, `cs_v` stores these values. In the case of a snapshot raster, the value will point to itself, indicating that it is a snapshot. For this reason, `s_bv` is not necessary to indicate snapshot selected rasters. `s_bv` can be computed

Figure 4.1: Clustering application example, with its respective $\texttt{s\_bv}$ and $\texttt{cs\_v}$ arrays for each snapshot selection. The gray cells in the list of rasters represent the rasters belonging to a first cluster, the white cells represent the rasters belonging to a second cluster, and the black cells represent the rasters belonging to a third raster. $\texttt{s\_bv}$ is only represented for illustrative reasons. $\texttt{s\_bv}$ can be computed as $\texttt{s\_bv}[i] = \texttt{cs\_v}[i] == i$.

from $\texttt{cs\_v}$ (i.e. $\texttt{s\_bv}[i] = \texttt{cs\_v}[i] == i$). Therefore, the storage overhead associated with storing snapshot references is minimal and does not significantly impact the overall size of the representation.

In the rest of this section, we describe the relevant configuration steps required for the application of clustering. First, Section 4.4.1 presents the distance measures applied in this study. Then, Section 4.4.2 describes the selection of the number of clusters. Finally, Section 4.4.3 details the selection of a raster centroid for each cluster, which is then represented as a snapshot whereas all the other rasters in the cluster are represented as logs with respect to such a snapshot.

4.4.1 Distance measures

The choice of a suitable distance measure is crucial to compare a set of rasters, as it must be sensitive to the differences between any pair of rasters. If the two rasters are identical, meaning that the values of all their cells are the same, the distance should be zero. As the differences between the rasters increases, the distance metric should also increase accordingly. To define a distance measure that effectively captures the differences between two rasters, we consider two criteria: (1) the number of cells that differ between the rasters and (2) the magnitude of the differences between those

cells. By incorporating both criteria into the distance measure, we aim to establish a comprehensive measure that appropriately accounts for the variations between rasters.

We applied two distance measures based on the Hamming distance [71]. The *Normalized Hamming distance* (NHD) [111, 110], a variant of the Hamming distance, is commonly used in decision making process, but its most basic application is to compare strings of equal length by counting the differing symbols. In the context of raster data, this distance measure can be adapted to quantify the dissimilarity between two rasters by counting the number of differing cells. Let M and N be two matrices with $|M|$ and $|N|$ cells, respectively (where $|M| = |N|$), and let m_i and n_i denote the value of a cell in M and N, respectively. Equation 4.2 presents the formula, where the count of differing cells is divided by the total number of cells in the raster. The resulting value ranges between 0 and 1, with values closer to 0 indicating more significant similarity between the compared rasters.

$$H = \frac{\sum_{0 \leq i < |M|} (m_i \neq n_i)}{|M|} \tag{4.2}$$

Where H is the Normalized Hamming distance (NHD), $m_i \in M$, $n_i \in N$, and m_i and n_i are in the same position from its respective rasters.

The *Weighted Hamming Distance* (WHD) [111, 110] is a second variant of the Hamming distance. It calculates the average of the absolute differences between all cells of two compared rasters (see Equation 4.3). The resulting value measures the dissimilarity between the rasters, with a larger H_w indicating more significant differences between them.

$$H_w = \frac{\sum_{0 \leq i < |M|} |m_i - n_i|}{|M|} \tag{4.3}$$

Where H_w is the Weighted Hamming Distance (WHD).

The *Combined Hamming Distance* (CHD) is a distance measure that weighs the NHD and WHD. The formula for CHD can be expressed as shown in Equation 4.4.

$$H_c = H \times H_w \tag{4.4}$$

Where H_c is the Combined Hamming Distance (CHD).

Distance	$[2,2]$ **and** $[4,4]$	$[2,2]$ **and** $[6,2]$
H_w	2	2
H_c	2	1

Table 4.1: Example of computation of H_w and H_c distances measures over 1×2 matrices

Table 4.1 shows an example showing the differences of applying these distance measures presented. In this example, the distances computed using H_w yield the same result for both computations, as the weight values are identical. However, H_c produces different results. This discrepancy arises because the two matrices in the first computation differ in two cells, while in the second computation, they differ in only one cell. H_c can capture the difference between rasters based on the number of cells that differ and the average variation of cell values.

In Section 4.5, we evaluate two distance measures, the Weighted Hamming Distance (WHD) and the Combined Hamming distance (CHD) to analyze the impact of the original Hamming distance on the weighted changes.

4.4.2 Selection of the number of clusters

Selecting the number of clusters is a crucial aspect of successful clustering. This study employed two strategies to determine a suitable number of clusters for the different clustering techniques.

The first strategy applies the Silhouette index described in Section 4.2.1. The Silhouette index helps to identify a suitable number of clusters, ensuring better grouping of the temporal clusters. The experimental results in [157] demonstrate that the Silhouette index presents a very good performance. The clustering technique is applied for each possible value of k between 2 and the total number of rasters. Next, the Silhouette index is computed based on the clustering performed. The value of k that produces the highest Silhouette index indicates that k is an appropriate value for the number of clusters.

For comparison purposes, a second straightforward strategy is proposed, which involves selecting the number of snapshots generated by constructing the Heuristic

T-k^2-raster. In the example shown in Figure 3.1, the number of clusters is 4, representing the number of clusters defined by the T-k^2-raster, as each snapshot represents a separate cluster. This strategy aims to analyze the performance of clustering techniques after using different clusters corresponding to the number of snapshots in their respective T-k^2-raster.

4.4.3 Selection of snapshots to represent a cluster

The final step involves the selection of the raster snapshot within each cluster. This selection enables the structure to consider the remaining rasters in the cluster as raster logs associated with the snapshot of their respective cluster.

Selecting the ideal snapshot within each cluster is crucial to minimize the final size of the resulting structure. In this study we considered two strategies for selecting a good snapshot within each cluster: First Raster Selection and Best Raster Selection.

In the first strategy, the first raster (in time-order) in each group is selected as the cluster snapshot. The strategy maintains similarities concerning the Heuristic T-k^2-raster heuristic. In both cases, the first raster that the cluster represents is defined as a snapshot and the other rasters, temporarily located in the future, are referenced to a snapshot raster temporarily located in the past.

The second strategy selects the raster within the cluster that reduces the sum of the distances to all the other rasters in the cluster, where the distance value corresponds to the distance definition used in the clustering step.

Figure 4.1 illustrates the application of clustering on a raster time series using a T-k^2-raster structure. In this scenario, `cs_v` is a relevant structure for identifying the snapshot rasters and establishing the corresponding mappings between logs and their respective snapshot references. It should be noted that, according to the snapshot raster selection, values `s_bv` and `cs_v` would differ, as is the case of the example given. For example, in the case of first raster snapshot selection, the rasters snapshot selected are M_1, M_2 and M_3, while for best raster snapshot selection, rasters snapshot are M_2, M_6 and M_7.

4.5 Experimental Results and Discussion

This section presents the experimental results that evaluate the different strategies described in the previous sections. We evaluate the approximation that disregards temporal order and uses clustering to potentially achieve better groupings of similar rasters.

4.5.1 Experimental framework

Server configuration: All the experiments were run on a dedicated Intel® Xeon® Gold 5320T CPU clocked at 2.30 GHz (40 physical cores) with cache sizes 1.9 MB (L1d), 1.3 MB (L1i), 50 MB (L2), and 60 MB (L3), and 252 GB of RAM. The operating system was Debian 11 with kernel 5.10.0-13-amd64. The C++ code was compiled with gcc version 10.2.1 and the -O3 optimizations. The Python code was executed with version 3.9.2.

Implementation code: The distance measures presented in Section 4.4.1 were implemented using the C++ programming language. The clustering algorithms were implemented in Python using the popular `scikit-learn` library [91]. For K-means (or K-medoids as used in the library), the initialization method of `k-medoids++` was used to select each cluster's representatives or centroids. The `scikit-learn` library also provides a function for computing the Silhouette index for each applied clustering method [141].

All the code was made available at a public repository[1], including the implementation of the clustering techniques discussed in Section 4.2.1, and the two snapshot selection strategies discussed in Section 4.4.3.

Datasets: In this study, we use real-world, synthetic, and semi-synthetic datasets[2]. Regarding the real datasets, we used the NLDAS-2 collection. This collection was obtained from [166] and it was also used in the experimental process of [153]. This

[1] `https://gitlab.com/mmunocan/clustering/`

[2] The data underlying this chapter are available in `https://figshare.com/s/5ad53959f8eed8a83f83`

collection is a product of the North American Land Data Assimilation System (NLDAS). It includes information of precipitation and flows across North America from 1979 up to the present, such as surface temperature, humidity, and radiation, among other variables. These raster time series have an hourly time resolution and a spatial resolution of $1/8$ degrees. The specific dataset used in our experiments correspond to the time period from January to December 2018.

Dataset	Minimum value	Maximum value	Unique values	Average value	Standard deviation
APCP	-1	10258	3268	5.43	38.08
CONVfrac	-1	100	102	1.12	9.87
DLWRF	9919	47911	24167	22147.16	8254.10
PEVAP	-79	185	263	-10.05	40.21
SPFH	-1	2	4	-0.19	0.48

Table 4.2: Main statistics of real-world raster times series

Table 4.2 presents detailed information regarding the datasets. All datasets presents 224 rows, 464 columns and 2664 rasters. APCP represents accumulated precipitation [mm], CONVfrac represents fraction of total precipitation that is convective, DLWRF represents downward longwave radiation flux [W/m^2], PEVAP represents potential evaporation [kg/m^2], and SPFH represents specific humidity [kg/kg].

We also use synthetic and semi-synthetic datasets to provide more insight on the evaluation of the clustering approach. Specifically, these datasets are used to test the hypothesis commented in Section 4.4 that indicates that the proposed clustering strategy helps to reduce the space usage in raster time series that exhibit a cyclic structure, i.e. that the same rasters repeat, either fully or partially, over time. For this, we generated cyclic temporal rasters where a small set of rasters are repeated until the total number of expected rasters is achieved. All the datasets generated have 224 rows, 464 cols, and 2664 rasters for analogy with the real-world datasets described above. Both the synthetic and the semi-synthetic datasets are generated analogously using the two strategies described below. Hence, the only difference between them lies in that the rasters forming the seed set S in synthetic datasets are artificially generated, whereas in semi-synthetic datasets, they are real rasters. The two strategies that we

apply to generate synthetic and semi-synthetic datasets with different characteristics are as follows:

- *Regular cyclic datasets*: Given a seed set S of rasters, a new dataset is generated by selecting the first x rasters of the set, where $x \in \{12, 24, 36, 48\}$ and repeating them (keeping their original order) until the final size of the dataset is achieved. Note that all the cycles inside the temporal raster have the same size. For example, if $S = \{A, B, C, D, E, F, G, \ldots\}$, $x = 3$ and the expected dataset size is 3, the generated dataset would be $\{A, B, C, A, B, C, A, B, C\}$.

- *Irregular cyclic datasets*: Given a seed set S of rasters, a new dataset is generated by iteratively selecting the first up to x rasters of the set, where $x \in \{12, 24, 36, 48\}$. On each iteration, the number of selected rasters is chosen randomly from the range $[1..x]$. The selected rasters are concatenated until the final size of the dataset is achieved. In this case, the cycles inside each temporal raster might not all have the same size. Using the same parameters of the previous example, a possible generated dataset could be $\{A, B, A, B, C, A, A, B, C\}$.

Synthetic datasets are created by initially generating the seed set of rasters S with randomly assigned cell values. The values, ranging from 0 to 100, are randomly distributed across the cells following a uniform distribution. These datasets replicate scenarios devoid of spatial or temporal localities.

Semi-synthetic datasets are formed using the first x rasters extracted from specific real-world datasets. In particular, we selected `APCP` and `CONVfrac` datasets. These datasets emulate scenarios characterized by spatial and temporal locality within a cyclic context.

Table 4.3 presents the characterization of the synthetic (Irregular Cycle) and semi-synthetic (Irregular CONVfrac and Irregular APCP) datasets generated. The table presents the number of cycles, the average cycle size, and the standard deviation of cycle size. We only present the information about irregular cycles because each generated dataset contains cycles of different random sizes, converting basic information such as the number of cycles and the average cycle size into unpredictable values before the data generation. In the case of the regular cycles datasets, due to all cycles

Dataset	Maximal cycle size	Number of cycles	Average cycle size	Standard deviation
Irregular Cycle	12	408	6.52	3.40
Irregular Cycle	24	216	12.32	6.90
Irregular Cycle	36	145	18.21	10.37
Irregular Cycle	48	108	24.48	13.87
Irregular `CONVfrac`	12	415	6.40	3.46
Irregular `CONVfrac`	24	214	12.39	6.81
Irregular `CONVfrac`	36	146	18.24	10.27
Irregular `CONVfrac`	48	109	24.59	13.64
Irregular `APCP`	12	422	6.30	3.46
Irregular `APCP`	24	218	12.15	6.88
Irregular `APCP`	36	138	19.24	10.46
Irregular `APCP`	48	110	23.98	13.82

Table 4.3: Main statistics of synthetic datasets with irregular cycles.

inside these datasets being the same size, it is easy to determine, for example, the total number of cycles generated. For example, in a dataset with regular cycles of length 12, `cycle12`, there are $2664/12 = 222$ cycles since the total number of raster is fixed to 2664.

Table 4.4 presents different measures that help to characterize the employed datasets. The *Moran Index* column [114] shows the average result of the Moran Index computed for each raster in the dataset. A value near 1 shows a high spatial autocorrelation or spatial locality, a value near 0 indicates a random distribution, and a value near -1 displays a perfect dispersion. This last value was absent in our results because we do not evaluate datasets with perfect dispersion. The *% of variation* column presents the average result of the percentage of variation comparing each raster with its predecessor. More precisely, the percentage of variation is the number of cells that differs comparing two rasters divided by the total number of cells inside a raster. A percentage close to 0 presents a high similitude between contiguous rasters or temporal locality.

Dataset	Moran Index	% of variation	Cyclic distance	Cyclic % of variation	% negatives distances
APCP	0.87	10.60	1.03	7.23	49.23
CONVfrac	0.81	1.85	4.19	0.86	51.28
DLWRF	0.97	40.71	1.00	22.39	51.20
PEVAP	0.97	19.56	1.03	0.02	66.64
SPFH	0.96	0.26	3.06	0.23	49.25
cycle12	0.00	99.00	12	0.00	99.55
cycle24	0.00	99.00	24	0.00	99.10
cycle36	0.00	99.00	36	0.00	98.65
cycle48	0.00	99.00	48	0.00	98.20
irre_cycle12	0.00	97.79	8.77	0.00	54.02
irre_cycle24	0.00	98.72	17.55	0.00	51.42
irre_cycle36	0.00	98.89	26.07	0.00	52.78
irre_cycle48	0.00	98.91	34.88	0.00	50.29
CONVfrac12	0.69	0.17	12	0.00	99.55
CONVfrac24	0.66	0.20	24	0.00	99.10
CONVfrac36	0.66	0.15	36	0.00	98.65
CONVfrac48	0.64	0.14	48	0.00	98.20
irre_CONVfrac12	0.72	0.19	8.71	0.00	53.72
irre_CONVfrac24	0.69	0.21	17.65	0.00	51.43
irre_CONVfrac36	0.67	0.19	25.98	0.00	51.77
irre_CONVfrac48	0.66	0.17	34.19	0.00	50.68
APCP12	0.79	4.96	12	0.00	99.55
APCP24	0.76	4.48	24	0.00	99.10
APCP36	0.77	3.84	36	0.00	98.65
APCP48	0.76	4.08	48	0.00	98.20
irre_APCP12	0.80	5.70	8.76	0.00	54.02
irre_APCP24	0.78	5.02	17.32	0.00	51.19
irre_APCP36	0.77	4.44	26.37	0.00	50.55
irre_APCP48	0.77	4.27	35.20	0.00	50.16

Table 4.4: Detailed description of the raster times series considered in the experiments. The *Moran Index* column shows the average result of the Moran Index computed for each raster in the dataset. The *% of variation* column presents the average result of the percentage of variation comparing each raster with its predecessor. The *cyclic distance* column shows the average distance between each raster and its most similar raster. The *cyclic % of variation* column shows the average variation between each raster and its most similar raster. The *% of negative distances* column shows the proportion of rasters whose most similar raster is located before it.

The last three columns measure the cyclicity of the datasets. The *cyclic distance*

column shows the average distance between each raster and its most similar raster, namely the raster with the smallest percentage of variation. The most similar raster can be forward or backward in the time serie. The *cyclic % of variation* column shows the average variation between each raster and its most similar raster. This value differs from the percentage of variation because the latter compares each raster with its preceding raster, which is not necessarily the most similar. Finally, the *% of negative distances* column shows the proportion of rasters whose most similar raster is located before it.

Baseline: For our baseline, we used the original implementation of the T-k^2-raster library.[3] This library includes the implementation of all variants of T-k^2-raster and k^2-raster. The implementation of the Heuristic T-k^2-raster, used as the principal baseline, represents each raster using the variant k^2_H-raster described in Section 2.2.1.

We compared the clustering-based method with other existing methods for representing a raster time series. Specifically, we compared our approach with the original Heuristic T-k^2-raster, as well as with an independent collection of k^2-raster (or k^2-raster Collection) and an independent collection of k^2_H-raster (or k^2_H-raster Collection). This analysis aimed to evaluate the performance and efficiency of our proposed method against these established techniques.

k^2-raster configuration: In order to construct the underlying k^2-raster, it is relevant to select the value of the four parameters, namely k_1, k_2, n_1, and l, described in Section 2.2.1. In this study, we select four combinations to determine the best parameter values. We decided to extend the configurations to improve the compression of each dataset and to make an effective comparison between the original structures and the clustering application. These four configurations were applied to construct the Heuristic T-k^2-raster, the independent collections, and the T-k^2-raster with clustering.

In order to compare each structure to obtain the best result, it is crucial to prepare the structure using the configuration that generates the smallest representation size of the raster time series. In this study, we chose the configuration that achieves the

[3] `https://gitlab.lbd.org.es/fsilva/k2-raster`

smallest representation size of the raster time series for each dataset and structure evaluated. We can obtain the most efficient representations by carefully selecting the parameters and configurations for each structure. This selection ensures the comparison of the structures under optimal conditions, enabling accurate and meaningful comparisons.

Dataset	**k^2-raster Collection**	**k^2_H-raster Collection**	**Heuristic T-k^2-raster**
APCP	4-2-3-1, 4-2-3-2	4-2-3-1	4-2-3-2
CONVfrac	4-2-3-1, 4-2-3-2	4-2-3-1	4-2-4-1
DLWRF	4-2-4-1, 4-2-4-2	4-2-4-2	4-2-4-1
PEVAP	4-2-3-1, 4-2-3-2	4-2-3-1	4-2-3-1
SPFH	4-2-4-1	4-2-4-1	4-2-3-2, 4-2-4-2

Table 4.5: Best configuration chosen for each structured compared. Each configuration is presented as k_1-k_2-n_1-l, where k_1 and k_2 represent the two k values used, n_1 represent the number of levels that use k_1, and l defines the number of levels that use the entropy-based dictionary.

In the case of the k^2-raster Collection, k^2_H-raster Collection, and Heuristic T-k^2-raster, we chose the best values of k_1, k_2, n_1 and l. All k^2-rasters use two k values: a k_1 that subdivides the first n_1 levels and a k_2 that subdivides the rest of the levels. Additionally, the structures use an entropy-based dictionary to represent the last l levels. For all combinations, the values of k_1 and k_2 were kept constant at 4 and 2, respectively. For n_1 and l, we select the best value between 3 and 4 for n_1 and between 1 and 2 for l. Table 4.5 shows the selected configurations for each representation and dataset. Each configuration is presented as k_1-k_2-n_1-l. For structures with two configurations, we choose the first option for queries evaluation.

4.5.2 Evaluating the application of clustering to T-k^2-raster

This section presents a comparative analysis of the implementation of the clustering technique versus k^2-raster Collection, k^2_H-raster Collection, and Heuristic T-k^2-raster. The main objective is to compare the effects of clustering against the baseline previously described, particularly the heuristic strategy. The clustering strategy should

expand the range of available rasters snapshots options for each raster log, providing better alternatives that contribute to reducing the size of the data structure. In this context, we include the synthetic and semi-synthetic datasets that present a cyclic temporal behavior in this analysis. Those datasets may take advantage of the additional alternatives of snapshot selection that clustering offers.

Dataset	**Configuration**			
APCP	Kmedoids	WHD	Best Centroid	4-2-3-2
CONVfrac	Hierarchical	WHD	Best Centroid	4-2-3-2
CONVfrac	Kmedoids	CHD	Best Centroid	4-2-4-1
DLWRF	Hierarchical/Kmedoids	WHD/CHD	Best Centroid	4-2-4-1
PEVAP	Hierarchical/Kmedoids	WHD/CHD	First/Best Centroid	4-2-3-1
SPFH	Kmedoids	WHD	Best Centroid	4-2-3-2/4-2-4-2

Table 4.6: Best configuration chosen to represent the Heuristic T-k^2-raster based on clustering. The last column shows the k^2-raster configuration presented as k_1-k_2-n_1-l, where k_1 and k_2 represent the two k values used, n_1 represent the number of levels that use k_1, and l defines the number of levels that use the entropy-based dictionary.

Parameters tuning: For clustering applications, the first step is to precompute the distance matrix. It corresponds to a square matrix that stores the distance values of all the rasters with each other. With the precomputed distance matrix, the second step is to apply the corresponding clustering technique and, later, the snapshot selection. The results correspond to the `cs_v` vector that the T-k^2-raster adapted required as an input. Tables 4.5 and 4.6 show the configuration used for each data structure. The k^2-raster configuration is presented as k_1-k_2-n_1-l. For structures with two or more configurations, we choose the first option for queries evaluation. All configurations use the number of clusters presented in Table 4.7.

Table 4.7 presents the number of clusters selected for this experimental process. The corresponding Silhouette value is shown in parentheses. The first strategy selection is based on the Silhouette Index result, while the second is based on the number of snapshots defined by the heuristic.

Dataset	Hierarchical WHD	Hierarchical CHD	Kmedoids WHD	Kmedoids CHD	Configuration k_1-k_2-n_1-l			
					4-2-3-1	4-2-3-2	4-2-4-1	4-2-4-2
APCP	2 (0.092)	2 (0.428)	12 (-0.045)	10 (-0.577)	889	888	889	858
CONVfrac	2 (0.407)	4 (0.340)	2 (0.254)	15 (0.294)	740	633	685	462
DLWRF	888 (0.999)	888 (0.999)	888 (0.999)	888 (0.999)	888	888	888	888
PEVAP	889 (0.999)	889 (0.999)	889 (0.999)	889 (0.999)	889	889	889	889
SPFH	2 (0.515)	2 (0.599)	2 (0.560)	2 (0.573)	24	14	14	26

Table 4.7: Selected number of clusters for clustering technique and for the Heuristic. In parentheses shows the corresponding Silhouette value. k_1-k_2-n_1-l values represent the k^2-raster configuration, where k_1 and k_2 represent the two k values used, n_1 represent the number of levels that use k_1, and l defines the number of levels that use the entropy-based dictionary.

Dataset	Size [MB]				Percentage of improvement (%)		
	k^2-raster Collection	k_H^2-raster Collection	Heuristic T-k^2-raster	Best cluster result	k^2-raster Collection	k_H^2-raster Collection	Heuristic T-k^2-raster
APCP	55.74	54.95	43.74	54.76	1.79	0.38	-25.15
CONVfrac	19.07	19.67	12.87	16.13	16.20	18.76	-24.16
DLWRF	450.54	365.50	298.45	283.19	37.14	22.52	5.11
PEVAP	88.14	82.12	27.66	27.68	68.60	66.29	-0.07
SPFH	7.14	7.66	5.35	5.38	24.51	29.63	-0.75

Table 4.8: Structure sizes using the alternative representations for each dataset.

Space evaluation: Table 4.8 presents the size of the data structures compared with the corresponding percentage of improvement. To determine the percentage of improvement, the formula $\frac{bs-cl}{bs} \times 100$ was used, where bs denotes the baseline size of the structure and cl represents the size of the T-k^2-raster with clustering. When comparing the clustering results with k^2-raster Collection and k_H^2-raster Collection, the clustering improvement percentage is between 0.38% and 68.60%. This suggest that clustering techniques for snapshot selection provides better compression results for rasters time series than both k^2-raster and k_H^2-raster collection which do not use snapshots selection.

Conversely, the improvement decreases when comparing the cluster results with the Heuristic T-k^2-raster. APCP and CONVfrac datasets present a negative improvement where the clustering technique increases the structure size. Table 4.7 explains the results by a low Silhouette Index. A low Silhouette Index indicates the difficulty of the clustering technique in finding an optimal cluster number.

SPFH and PEVAP achieve competitive results, with clustering sizes comparable to

those obtained in the Heuristic T-k^2-raster. After a detailed datasets revision, we discovered that the reason is different for both datasets. In the case of SPFH, the outcome can be attributed to the dataset's low number of distinct values, resulting in minimal variation between values. For PEVAP, the snapshot and cluster selections are the same for clustering and the heuristic strategies. In addition, the Silhouette Index for PEVAP indicates an high value (close to 1) according to Table 4.7.

DLWRF is the only case in which the clustering technique reduces the space usage compared with the heuristic. The Silhouette Index also presents a value close to 1 (see Table 4.7). After a manual inspection of the generated data structures, we discovered that although both strategies select the same clusters, inside each cluster they select different snapshots. Specifically, both techniques select groups of three contiguous rasters as clusters, however, while the heuristic selects the first raster in each cluster as a snapshot, clustering always selects the second raster because it uses the Best Raster Selection strategy explained in Section 4.4.3. The heuristic approach, by design, systematically selects the first raster within each cluster as the snapshot without considering factors such as its distance to other rasters. Conversely, employing clustering with the Best Raster Selection strategy provides increased adaptability. This method considers the distance between the chosen snapshot and the remaining rasters within the cluster. Consequently, it enables the identification of a more suitable alternative snapshot that effectively minimizes the structure size compared to the heuristic method. This difference allows the clustering technique to obtain an advantage with respect to the heuristic.

We can observe some correspondences if we compare the results obtained in Table 4.8 with the measures presented in Table 4.4. On the one hand, APCP, CONVfrac, and SPFH present a low difference between the percentage of variation and its cyclic version. Instead, DLWRF and PEVAP present a significant difference in both percentages of variation. These results indicate that if the raster variation does not significantly differ comparing a raster with its similar raster versus the last raster, T-k^2-raster structure would not obtain an improvement after applying the clustering technique.

Another interesting result is the high percentage of negative distances on PEVAP dataset compared with the rest of the real-world datasets. Including the cyclic distance

near 1, we can infer that for each raster, the most similar raster is usually located immediately before it. This scenario could be optimal for the heuristic strategy, such as the case of PEVAP.

Dataset	Get Cell		Get values window		Get cells by value	
	Heuristic T-k^2-raster	Cluster result	Heuristic T-k^2-raster	Cluster result	Heuristic T-k^2-raster	Cluster result
APCP	0.37	0.30	12.81	13.55	34,320.40	32,370.20
CONVfrac	0.23	0.18	8.12	7.63	377.55	356.48
DLWRF	0.78	0.80	26.18	26.87	101.42	106.98
PEVAP	0.40	0.41	18.52	19.23	78.86	81.49
SPFH	0.13	0.15	6.15	6.91	17.70	19.76

Table 4.9: Query time results for Heuristic T-k^2-raster and clustering. Get cell (in μs/query, 100x100,000 queries), Get values window (in μs/cell, 100x100 queries), Get cells by value (in μs/cell, 100x100 queries)

Query time evaluation: To compare query time, we evaluated the three queries implemented in [153]: *Get Cell* (retrieves the value of the cell), *Get values window* (retrieves all the cell values in a rectangular cuboid defined) and *Get cells by values* (retrieves all the cells inside a rectangular cuboid whose values are within a defined range). The results show that the cluster T-k^2-raster has a similar query time performance as the Heuristic T-k^2-raster. Applying the cluster in the T-k^2-raster does not significantly modify query time efficiency. Table 4.9 presents the query time results for the respective queries over real-world datasets.

Cyclic datasets results: Now we focus on the analysis regarding cyclic datasets. To check if the clustering technique can detect the optimal number of clusters, we calculate the Silhouette index for all the synthetic datasets and each possible value of k between 2 and the total number of temporal rasters. The objective was to identify the cluster size that maximized the Silhouette index, which indicates the optimal number of clusters. For all datasets, the computed number of clusters coincides with the cycle length of regular cycle datasets and the maximal cycle length for irregular cycle datasets. These results indicate the effectiveness of the clustering technique in

selecting the appropriate number of clusters and the subsequent snapshot selection process.

Table 4.10 presents the size of the structures and their corresponding percentage of improvement for cyclic datasets. To determine the percentage of improvement, the formula $\frac{bs-cl}{bs} \times 100$ was used, where bs denotes the baseline size of the structure and cl represents the size of the T-k^2-raster with clustering. We exclude `CONVfrac` datasets because it has similar results compared with `APCP` datasets. Comparing Heuristic T-k^2-raster with the best cluster result, we can observe a significant reduction in the data structure size by applying the clustering technique, especially in the synthetic datasets. Even for datasets with regular cycles, Heuristic T-k^2-raster presents a similar size that k_H^2-raster Collection, indicating that the heuristic selects all rasters as a snapshot, similar to a k_H^2-raster Collection. In the case of semi-synthetic datasets, the difference is less because these datasets present a high spatial locality and a more significant temporal locality than synthetic datasets.

Dataset	Size [MB]				Percentage of improvement (%)		
	k^2-raster Collection	k_H^2-raster Collection	Heuristic T-k^2-raster	Best cluster result	k^2-raster Collection	k_H^2-raster Collection	Heuristic T-k^2-raster
Cycle12	345.41	260.81	260.81	1.59	99.54	99.39	99.39
Cycle24	345.45	260.83	260.83	2.77	99.20	98.94	98.94
Cycle36	345.43	260.82	260.82	3.94	98.86	98.49	98.49
Cycle48	345.43	260.81	260.81	5.11	98.52	98.04	98.04
Irregular_Cycle12	345.45	260.80	257.61	1.59	99.54	99.39	99.38
Irregular_Cycle24	345.42	260.81	260.09	2.77	99.20	98.94	98.93
Irregular_Cycle36	345.46	260.82	260.43	3.94	98.86	98.49	98.49
Irregular_Cycle48	345.42	260.82	260.58	5.02	98.55	98.08	98.07
APCP12	25.25	27.43	18.41	0.54	97.86	98.03	97.07
APCP24	24.19	26.39	16.92	0.66	97.27	97.50	96.10
APCP36	22.31	24.31	14.89	0.74	96.68	96.96	95.03
APCP48	23.25	25.20	15.31	0.87	96.26	96.55	94.32
Irregular_APCP12	26.18	28.59	18.53	0.54	97.94	98.11	97.09
Irregular_APCP24	25.19	27.44	17.74	0.66	97.38	97.59	96.28
Irregular_APCP36	25.90	26.01	16.50	0.74	97.14	97.15	95.52
Irregular_APCP48	23.44	25.47	15.93	0.87	96.29	96.58	94.54

Table 4.10: Structure sizes using the alternative representations for each dataset.

4.6 Conclusions and Future Work

In this chapter, we delve into the study of efficient space representation for raster time series based on T-k^2-raster. This structure is a Compact Data Structure (CDS) that classifies each raster in snapshots and logs, in which snapshots serve as references to logs. It is important to recall that only the most recently generated snapshot, following the time-order, can be referenced by a log. The heuristic presented in Heuristic T-k^2-raster selects a suitable subset of snapshots in order to reduce the space of the representation.

Subsequently, we studied the potential enhancements achievable through non-time-ordered variants, employing clustering techniques. In this approach, we group the rasters in the time series based on similarity using various measures derived from the Hamming distance. Next, for each cluster, one raster is selected as a snapshot, while all the others are encoded as logs with respect to the chosen snapshot. We use an array to identify, for each raster, its corresponding snapshot. Our experimental evaluation demonstrates that clustering can achieve an equivalent or improve compression performance for most of the tested datasets of the Heuristic T-k^2-raster while maintaining query support performance. We identify that clustering performs the best in datasets with rasters repeated in cycles.

As part of our future work, we plan to delve deeper into the application of clustering techniques. Specifically, we intend to investigate the implementation of DBSCAN [83] and explore an alternative approach to characterizing the rasters. This alternative approach involves extracting features from each raster, such as its width, height, the maximum/minimum/average values in the raster, and some indexes such as Moran [114], Geary [162], and Getis [132], and use the cosine distance to compute the similarity between the feature vectors that characterize each raster.

Chapter 5

Estimating the Compressibility of Raster Data

5.1 Introduction

In this chapter, we present a comprehensive evaluation of the compressibility of rasters and temporal rasters. We tackle this in two different ways: *i)* First, we transform rasters and temporal raster in 1D sequences using Space-Filling Curve (SFC) [1] and next compute well-known 1D compressibility measures [122] to estimate the compressibility of the rasters. We compare this estimation against 2D compressibility measures [31, 140], compressors, CDSs, and other related measures. *ii)* Second, we address the insensitivity of existing compressibility measures to the alphabet, a limitation uncovered during our study. Any modification to the alphabet that does not alter its size does not affect the outcome of these measures, unlike some compressors, which are indeed affected by alphabet variations. We propose $\delta_{\triangle}$, based on the 2D compressibility measure δ_{2D} [31, 140] that incorporates differences in alphabet values into its computation.

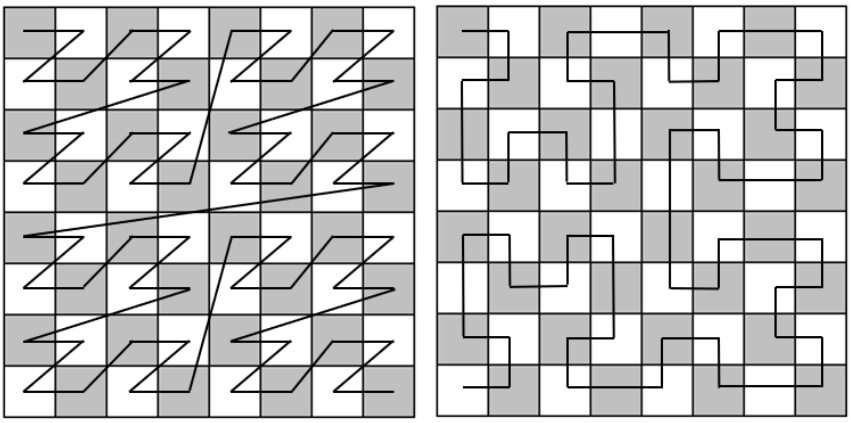
The structure of this chapter is as follows: Section 5.2 provides the necessary background to contextualize the study. Section 5.3 presents related work that introduces multidimensional measures. Section 5.4 elaborates on the proposals outlined in this study. Section 5.5 describes the experimental framework, and Section 5.6 discusses the results. Finally, Section 5.7 summarizes the main conclusions and proposes future work.

5.2 Background

This section presents the specific background necessary to understand our study. Section 5.2.1 introduces Space-Filling Curves (SFCs). Section 5.2.2 describes the compressibility measures proposed in the literature. Section 5.2.3 briefly reviews the compressors employed in this work. Finally, Section 5.2.4 presents the spatial autocorrelation measures used in our analysis.

5.2.1 Space-Filling Curve (SFC)

Most compressibility and repetitiveness measures considered in this study are designed for their application to sequences. Therefore, applying these measures to rasters necessitates a previous linearization process. A linearization function requires a *Space-Filling Curve* (SFC). We can see an SFC as a path that sequentially traverses each raster cell, effectively connecting them into a continuous thread. By stretching the thread, we will have a linearized raster, allowing the application of sequence-based measures.



(a) Z-Order Curve example (b) Hilbert Curve example

Figure 5.1: Space-Filling curves examples

There are different SFCs, and we have selected three distinct types for this study. The first of them is *Row Major Curve*. This SFC is a naive curve that traverses the raster cells row by row, moving from left to right across each row and proceeding from the top to the bottom of the raster. The second SFC is *Z-Order Curve* or Morton

Curve [115], which recursively traverses the space in a pattern resembling the letter Z. Figure 5.1a illustrates an application example of the SFC on an 8×8 raster. The third SFC is *Hilbert Curve* [73], traversing the space recursively but following a path that creates a tower-like structure. Figure 5.1b presents an application example of Hilbert Curve on a 8×8 raster.

Z-Order and Hilbert preserve better spatial locality [1]. This property ensures that cells that are close together in the original raster remain close in the resulting sequence after applying these SFCs. The preservation of spatial locality will significantly impact the compressibility measures applied to these sequences, potentially leading to observable differences compared to the Row Major curve results.

5.2.2 Compressibility measures

Compressibility measures aim to estimate the compressibility of sequences. Navarro [122] discusses several compressibility measures based on sequence repetitiveness. It is important to note that this definition of repetitiveness considers only exact repetitions without accounting for slight variations in values, which will be relevant to our work.

Now, we describe the compressibility measures presented in the literature:

Empirical entropy The classical measure of compressibility is the *Empirical Entropy* [160], derived from Shannon Entropy [147]. It estimates compressibility from the frequency of symbols within a sequence. Equation 5.1 defines the Empirical Entropy (H_0) of a sequence S, where Σ is its alphabet, n the sequence length, and n_a the frequency of symbol a. Example 1 [121] presents an example of the computation of this measure.

$$H_0(S) = \sum_{a \in \Sigma} \frac{n_a}{n} \log \frac{n}{n_a} \tag{5.1}$$

Example 1. *For* $S =$ `abracadabra`, $n_a = 5$, $n_b = 2$, $n_r = 2$, $n_c = 1$, $n_d = 1$, *and* $n = 11$, *yielding* $H_0(S) = \frac{5}{11} \log \frac{11}{5} + 2 \times \frac{2}{11} \log \frac{11}{2} + 2 \times \frac{1}{11} \log \frac{11}{1} \approx 2.040$.

There is a variant called *Empirical entropy of order* k (H_k), which is based on the frequency of each symbol considering the k previous symbols. Equation 5.2 defines

H_k, where Σ^k is the set of all k-length sequences over Σ, S_C the concatenation of the k symbols following each occurrence of C in S, and $n_C = |S_C|$. Example 2 [121] illustrates this measure.

$$H_k(S) = \sum_{C \in \Sigma^k} \frac{n_C}{n} H_0(S) \tag{5.2}$$

Example 2. *Let* $S =$`abracadabra$`*. To calculate* H_1*, we have:* S_a=`bcdb$`*,* S_b=`rr`*,* S_r=`aa`*,* S_c=`a` *and* S_d=`a`*. Then,* $H_0(S_a) \approx 1.922$*,* $H_0(S_b) = 0$*,* $H_0(S_r) = 0$*,* $H_0(S_c) = 0$ *and* $H_0(S_d) = 0$*. Finally, we have that* $H_1(S) = \frac{5}{11}H_0(S_a) + \frac{2}{11}H_0(S_b) + \frac{2}{11}H_0(S_r) + \frac{1}{11}H_0(S_c) + \frac{1}{11}H_0(S_d) \approx 0.874$*.*

Empirical entropy provides theoretical lower bounds of nH_0 and nH_k bits. The trend of empirical entropy is $H_{k+1} < H_k$, so the nH_k bound decreases as k increases. This observation might suggest that larger values of k lead to reduce space size and higher compression. However, this interpretation is misleading because the bound does not account for the extra space needed to reconstruct the original sequence. A better fit of the bound would be $nH_k + o(n)$ bits [121].

Number of runs Another classical measure is the *number of runs* (r_S) [69]. A run is a maximal subsequence of identical symbols. This measure counts the minimum number of runs in S. The closer identical symbols appear, the fewer runs the sequence contains. Each run can be represented by the symbol it contains and the size of the run. Using *Run-Length Encoding* (RLE), a sequence of length n can be compressed in $r_S(\log \sigma + \log n)$ bits, where $\sigma = |\Sigma|$.

Measures z This measure derives from the `Lempel Ziv` compressor [102], which exploits repetitiveness of a sequence. The compressor scans the sequence from left to right, segmenting it into phrases and encoding each in a constant space. The version considered for measuring repetitiveness is LZ77 [169]. The measure z is the number of phrases the compressor generates.

Let $S[1..n]$ be a sequence. The compressor operates as follows:

1. Let i be the position where a new phrase begins. The compressor searches for the largest prefix $S[i..j]$ within $S[i..n]$ that appears fully or partially in $S[1..i-1]$.
2. If $j \geq i$, the new phrase is $S[i..j]$, its source is $S[i'..j']$, where $i' < i$. If $S[i]$ has not appeared before, $S[i]$ is a new phrase based on a new symbol.
3. We change the value of i to $j+1$ if the generated phrase has a source or $i+1$ if the generated phrase is a new symbol.

Example 3 [122] presents an example of the Lempel Ziv computation and the z measure.

Example 3. *Let $S =$`alabaralalabarda$`. Figure 5.2 shows its partition into 11 phrases according to the Lempel–Ziv parsing ($z = 11$). Phrases marked in gray correspond to explicit symbols, and the arrows indicate the source position of each non-explicit phrase.*

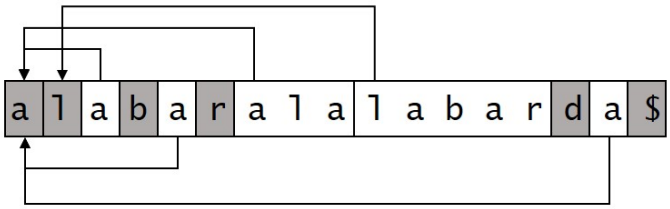


Figure 5.2: Lempel–Ziv parsing of $S =$`alabaralalabarda$`

There are two variants of the z measure. The first, z_{no}, requires that $j' < i$, namely, the repeated subsequence must be fully contained within $S[1, i-1]$ [159]. The second variant, z_{end}, requires that the end of every source $S[i', j']$ matches the end of a phrase [92]. Consequently, it holds that $z \leq z_{no} \leq z_{end}$.

Lempel Ziv allows the compression of a sequence in $O(z)$ space and in linear time [58]. The Block Tree [20] is a CDS that compresses a sequence in $O(z \log \frac{n}{z})$ space [122].

Measure g and g_{rl} The measure g denotes the size of the smallest *Context-Free Grammar* [85] that generates only S. The size of a grammar is the total length of the right-hand sides of all its production rules.

A variant is g_{rl}, which corresponds to the size of the smallest *Run-Length Context-Free Grammar* [127] that generates only S. This grammar includes rules of the form $A \to X^t$, where X is either a terminal or a non-terminal, and $t \geq 2$. Example 4 [122] presents an example of grammar and the computation of measure g.

Finding the smallest grammar is an NP-complete problem [159, 34]. Nevertheless, various heuristics approximate the smallest Context-Free Grammar, such as Re-Pair [100]. A data structure allows the representation of grammars in $O(g)$ space by compressing the parse tree [122]. Gagie *et al.* established the relationship $z \leq g_{rl} \leq g$ [63].

Example 4. *Let* $S = alabaralalabarda\$$. *A context-free grammar that generates only* S *contains the following rules:* $A \to al$, $B \to Aabar$, *and* $C \to BABda\$$. *The size of the grammar is* $g = 13$.

Measure c The measure c denotes the size of the smallest *Collage System* [84] that can generate only S.

A Collage System extends Context-Free Grammar rules with trimming operations and may include productions of the form $A \to B^{[t]}$ and $A \to^{[t]} B$. $B^{[t]}$ represents the last t symbols of the sequence generated from the nonterminal B, while ${}^{[t]}B$ represents the first t symbols of the sequence generated from the nonterminal B.

Example 5 [122] presents an example of a Collage System and the value of the measure c.

Example 5. *Let* $S = alabaralalabarda\$$. *A Collage System that generates only* S *contains the following rules:* $A \to al$, $B \to AAabar$, $B' \to^{[6]} B$ *and* $C \to B'Bda\$$. *The size of the grammar is* $c = 14$.

Computing the measure is an NP-complete problem [84]. Kida *et al.* demonstrate that $c \leq g_{rl}$ [84].

Measure b An extension of Lempel Ziv is the *Bidirectional Macro Scheme* (BMS) [159], in which the source subsequence may appear either to the left or right of the current position. This flexibility allows multiple BMS representations to exist for a given sequence. The measure b corresponds to the number of phrases the smallest BMS

generates. Computing the smallest BMS is an NP-hard problem; however, several heuristics provide approximate solutions [128, 142]. Since Lempel Ziv is a BMS, it follows that $b \leq z$ [63].

Example 6 [122] presents an example of a BMS and the measure's value b.

Example 6. *Let $S =$alabaralalabarda\$. Figure 5.3 shows the division of S into 10 phrases according to BMS ($b = 10$). Phrases marked in gray correspond to explicit symbols, and arrows indicate the source position of each phrase without explicit symbols.*

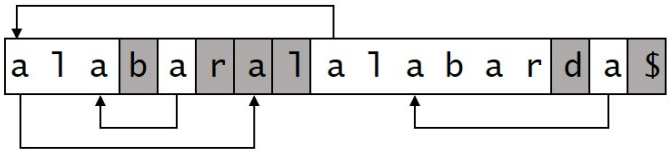


Figure 5.3: Division of $S =$alabaralalabarda\$ into BMS phrases

Measure r_{bwt} The measure is based on the *Burrows-Wheeler Transform* (BWT) [64]. The BWT is a reversible permutation of a sequence S, denoted as S^{bwt}. The measure r_{bwt} is the number of runs found in S^{bwt}.

Example 7 [122] presents the computation of the BWT and the corresponding value of r_{bwt}.

Example 7. *Let $S =$alabaralalabarda\$. Figure 5.4 represents the transformed sequence $S^{bwt} =$addl\$lrbbaaraaaaa. The number of runs is $r_{bwt} = 10$.*

The measure r_{bwt} can be computed in linear time [131, 17]. Using the Run-Length FM-index data structure, a sequence can be compressed using $O(r_{bwt})$ space [64]. Navarro *et al.* showed that $b = O(r_{bwt})$ because a BWT can induce a BMS of size $2r_{bwt}$ [123].

Measure e The e measure is based on the *Compact Direct Acyclic Word Graph* (CDAWG) [21]. The CDAWG is constructed from the Suffix Tree of a sequence [12]by

Figure 5.4: String S^{bwt} and ordered suffixes of $S =$`alabaralalabarda$`

merging all identical subtrees into a single directed acyclic graph. The measure is the sum of the number of nodes and edges in the graph. This measure can be computed in linear time. Belazzougui *et al.* demonstrated that $e = \Omega(\max(z, r_{bwt}))$ [19] and $e = \Omega(g)$ [18]. It is one of the weakest measures.

Measure v Measure v corresponds to the number of phrases produced by *lexicographical parsing* or *lex-parse* [123]. This method generates another Bidirectional Macro Scheme (BMS) derived from the Suffix Array (SA) [107]. The SA is a structure that helps maintain the lexicographic order of all possible suffixes of a sequence S.

Let $S[1, n]$ be a sequence. The lex-parse operates as follows:

1. Let i denote the starting position of a new phrase. Using the SA, search for $S[i..n]$ and identify $S[i'..n]$ as the preceding suffix in lexicographic order.
2. Determine the longest common prefix between $S[i..n]$ and $S[i'..n]$, denoted $S[i..j]$. This sequence is the new phrase. If $S[i] \neq S[i']$, then $S[i]$ is a new phrase based

on a new symbol.

3. Update i to $j + 1$ if the generated phrase has a source, or $i + 1$ if it represents a new explicit symbol.

Example 8 [122] presents an example of the lexicographical parsing application and the value of the measure v.

Example 8. *Let* $S =$`alabaralalabarda$`*. Figure 5.5 shows its partition into 11 phrases according to lex-parse (*$v = 11$*). Phrases highlighted in gray denote explicit symbols, and the arrows indicate the source position of each non-explicit phrase.*

Figure 5.4 shows the sorted suffixes of S*. We locate the suffix* `alalabarda$` *to obtain the phrase* `ala`*. Its previous suffix is* `alabarda$`*. The greatest common prefix between both is* `ala`*, so we identify a new phrase. We then continue processing the rest of the sequence, which is* `labarda$`*.*

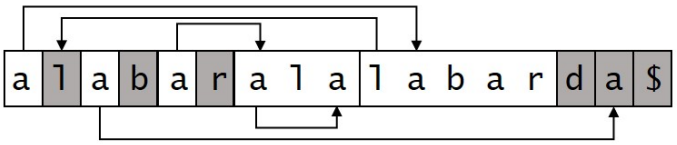


Figure 5.5: Lex-parse division of $S =$alabaralalabarda$

The measure v can be computed in linear time [80]. Since a lex-parse is a BMS, it holds that $b \leq v$ [122]. Navarro *et al*. showed that $v \leq g_{rl}$ and $v = O(r_{bwt})$ [123].

Measure γ The γ measure is based on the concept of *String Attractors* [81]. A string attractor is a set Γ of positions within a sequence S such that every subsequence $S[i..j]$ has at least one occurrence that contains a position from Γ. A sequence can contain multiple attractors. The γ measure is the size of the smallest attractor. Example 9 [122] presents the identification of an attractor and the computation of γ.

Example 9. *Let* $S =$`alabaralalabarda$`*. One possible attractor for this sequence is* $\Gamma = \{4, 6, 7, 8, 15, 17\}$*.*

Finding the smallest attractor is an NP-complete problem [81]. Since any BMS induces an attractor, it follows that $\gamma \leq b$ [122]. The Γ-tree data structure indexes a sequence in $O(\gamma \log(n/\gamma)$ space [81, 39].

Measure δ The δ measure is based on the concept of *String Complexity* S_k [39]. String complexity is the number of distinct subsequences of length k that occur in the sequence S. Equation 5.3 defines the $\delta(S)$ measure. The δ measure is defined as the maximum ratio between $S(k)$ and k over all k from 1 to n, as shown in Equation 5.3:

$$\delta = \max\left\{\frac{S(k)}{k}, 1 \leq k \leq n\right\} \tag{5.3}$$

Example 10 [122] presents an example of the computation of δ measure.

Example 10. *Let* $S =$`alabaralalabarda$`*. Table 5.1 calculates* $S(k)$ *and* $S(k)/k$*. The maximum value is* $k = 1$*, yielding* $\delta = 6$*.*

k	$S(k)$	$S(k)/k$
1	6	6.00
2	9	4.50
3	10	3.33
4	11	2.75
5	11	2.20
6	11	1.83
> 6	$17 - k + 1$	< 1.83

Table 5.1: $S(k)$ and $S(k)/k$ for the sequence $S =$`alabaralalabarda$`

The measure can be computed in linear time [80, 89]. Christiansen *et al.* showed that $\delta \leq \gamma$ [39]. By using the Block Tree data structure, it is possible to compress a sequence using $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space [88].

Measure l **and** ν The measures l and ν are recent additions to the set of repetitiveness measures [125]. Both are based on *morphisms*, which generalize grammars. Navarro and Urbina showed that $g_{rl} = o(l)$, $r_{bwt} = o(l)$, $\nu = O(l)$ and $\nu = O(b)$ [126].

Among all the repetitiveness measures, our study considers the following: H_k, r_S, z, v, r_{bwt}, g, and δ. Measures not included were excluded because their computation is an NP-problem, without an efficient heuristic approximating the result [122]. We computed the g measure applying the RePair Heuristic [100]. Although the measure e can be computed in linear time, we excluded it because it is the weakest measure [21].

Relation between measures The literature provides theoretical and empirical findings on the relationships between these measures. A notable empirical observation is $\delta < z \approx v < g < r < e$, where $<$ indicates a significant difference [122].

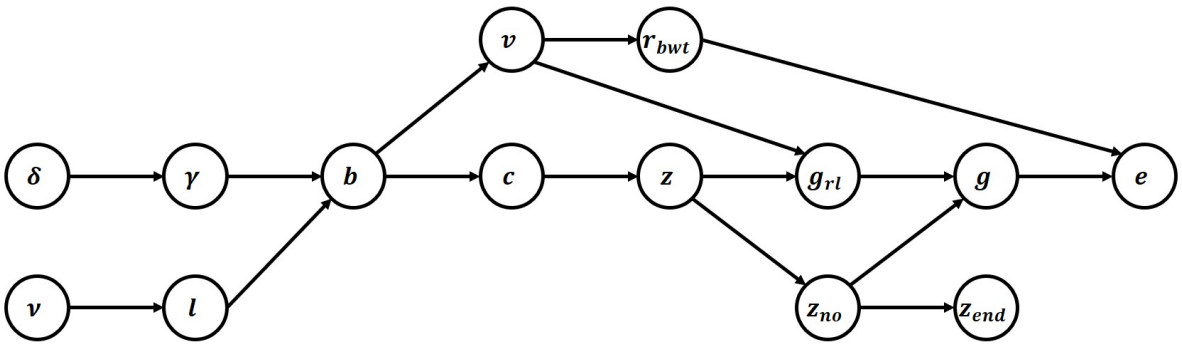


Figure 5.6: Relation between repetitiveness measures

Figure 5.6 presents the relation between repetitiveness measures. An arrow from measure X to measure Y indicates that $X = O(Y)$. Among all the measures, the asymptotically smallest measure computable in polynomial time is δ.

Giuliani *et al.* studied the influence of editing a single character in a sequence on measure r_{bwt} [67]. The results indicate that, for some string families, modifying a single character in the sequence significantly affects the measure. This result highlights the relevance of studying the sensitivity of compressibility measures to these changes when applied to rasters.

5.2.3 Compressors

We consider three flagship compressors: `bzip`[1], `gzip`[2], and NetCDF [101]. `bzip` and `gzip` are general-purpose compressors treating files as bit sequences. When processing a raster, both use a row-major traversal. `bzip` applies Burrows-Wheeler Transform (BWT) with Huffman coding [112], while `gzip` uses the Deflate algorithm [47], a combination of Lempel-Ziv and Huffman codi ng. Both allow adjustable compression levels from 1 (minimal) to 9 (maximal). NetCDF, on the other hand, was designed for n-dimensional array representation and uses the Deflate algorithm, similar to gzip. NetCDF segments the input raster into chunks and then compresses each chunk independently. This chunking strategy aims to reduce the decompression time during queries needing access to a specific region. Similar to previous cases, compression levels range from 0 (no compression) to 9 (maximum compression).

5.2.4 Spatial autocorrelation

Spatial autocorrelation estimates the degree of relationship between a raster's values and their spatial location. High spatial autocorrelation suggests nearby cells are more similar than distant cells, a property tied to spatial locality [65]. This study considers two spatial autocorrelation measures: *Moran's* I [114] and *Geary's* C [162]. Moran's I ranges from -1 to 1, with values near 1 indicating high spatial autocorrelation. Geary's C ranges from 0 upward, with values near 0 indicating high spatial autocorrelation. The two measures are inversely related: Moran's I increases as spatial autocorrelation rises, and Geary's C decreases.

5.3 Related Work

The literature presents two researches related to the application of a Space Filling Curve (SFC) and a compressibility measure on raster data [32, 52]. Carfagna *et al*. [32] presents a theoretical analysis of applying a SFC alongside a 1D compressor, studying the behavior of the measures δ, γ, and b when using row-major and Hilbert curves on

[1] `http://www.bzip.org/`
[2] `https://www.gzip.org/`

specific matrix families and comparing them with their respective 2D measures, which are explained later in this section. The principal conclusions of the authors are two: *i)* Given the Identity matrix I_{2^k} of size $2^k \times 2^k$, and its Hilbert linearization $hil(I_{2^k})$, they conclude that $b_{2D}(I_{2^k}) = O(1)$, $b(hil(I_{2^k})) \geq \gamma(hil(I_{2^k})) = \Omega(k)$. *ii)* Given the matrix M_n that appends to I_{n-1} a row of 0's at the bottom and a column of 1's at the right, and $rm(M_n)$ its row-major linearization, they conclude that $\delta_{2D}(M_n) = O(1)$ and $\delta(rm(M_n)) = \Omega(n)$. Reis and Kaster [52] introduce a CDS designed to compact a raster using a row-major curve combined with run-length compression. Their preliminary results indicate that this approach achieves efficient compression of rasters.

Our work differs from the literature's proposals because we conduct an experimental process to evaluate the measure's behavior with diverse factors, such as raster compressibility, spatial locality, and noise impact. Regarding Carfagna *et al*. [32], we share a similar study objective, although their approach is theoretical while ours is practical. As for Reis and Kaster [52], the objective differs, as they focus on proposing a specific CDS, using a particular SFC and compressor.

2D compressibility measures There are various extensions of repetitiveness measures within the 2D context. Among these measures, the literature presents: b_{2D}, g_{2D} with its variant g_{rl2D}, γ_{2D} and δ_{2D} [31, 140]. Computing b_{2D}, g_{2D} and γ_{2D} is an NP-Complete problem, while δ_{2D} can be computed in linear time using an *ISuffix Tree* [86].

$$\delta_{2D}(M) = \max\left\{\frac{S_{k\times k}(M)}{k^2}, 1 \leq k \leq \min(n,m)\right\} \tag{5.4}$$

Equation 5.4 defines the computation of δ_{2D} for a matrix M of dimensions $m \times n$. Let $S_{k\times k}(M)$ denote the number of distinct $k \times k$ submatrices within M. Similar to δ, this measure aims to maximize the ratio between $S_{k\times k}(M)$ and k^2.

The theoretical relation among the 2D measures states that $b_{2D} < \delta_{2D} \leq \gamma_{2D}$ and $b_{2D} \leq g_{rl2D} \leq g_{2D}$ [31, 140]. The smallest measure is b_{2D}, although its computation is NP-complete. Notably, δ_{2D} is the only feasible measure in this set, as it can be computed in polynomial time.

Extension to 3D measures In the 3D context, the measures γ_{3D} and δ_{3D} extend the definitions of γ and δ respectively [31]. Carfagna and Manzini conclude that $\delta_{3D} < \gamma_{3D}$ [31]. An alternative approach is to linearize a 3D matrix into a 1D sequence, enabling the application of various compressibility measures presented in Section 5.2. For instance, the Hilbert 3D curve [77] facilitates this transformation while preserving spatial and temporal locality in a raster time series.

5.4 Methodology

In this section, we present our two approaches for studying the compressibility of raster data. First, in Section 5.4.1, we explain a proposal based on the application of a Space-Filling Curve (SFC) to transform the 2D raster data into a 1D sequence, to then compute a 1D compressibility measure. Subsequently, in Section 5.4.2, we describe a second proposal that consists of a new 2D compressibility measure sensitive to the alphabet, named δ_{Δ}.

5.4.1 Application of one-dimensional compressibility measures on linearized rasters

As mentioned before, we compare three SFCs: row-major, Z-Order, and Hilbert curves. The 1D compressibility measures considered are H_k, r_S, z, v, g, r_{bwt}, and δ. All of them can be computed in linear time, except for g, for which we employ the RePair heuristic [100] that yields an approximate solution to the minimal grammar.

The repetitiveness of a sequence is not necessarily related to the locality or clustering of similar values within the sequence. While locality is related to a clustered distribution of similar symbols within the sequence, repetitiveness is related to repeated patterns throughout the sequence. For example, the 1D sequence `abcdabcdabcd` has a low locality, but a high repetitiveness. Conversely, the sequence `abcdefghi` has a high locality, but a low repetitiveness.

In addition to the combination of SFCs and 1D measures, we compute δ_{2D} [31, 140], a two-dimensional version of the δ measure. Additionally, we compare the compressibility measures against the k^2-raster [97, 98] and stand-alone compressors. With this

comparison, we aim to verify if compressibility measures can estimate the compressed size of the rasters.

As Tobler's First Law of Geography suggests, spatial locality is a fundamental property in raster compression [161]. For this analysis, we define the measure L as the average count of equal neighbors each raster cell has. Neighbors include the eight surrounding cells, except for edges and corners cells. This measure ranges from 0 to 8, with higher values indicating higher spatial locality. This measure is sensitive to variations in raster values; even small changes can significantly affect it.

$$R = \begin{bmatrix} 5 & 5 & 5 & 6 \\ 5 & 5 & 6 & 6 \\ 5 & 6 & 6 & 6 \\ 6 & 6 & 6 & 6 \end{bmatrix} \qquad L_c = \begin{bmatrix} 3 & 4 & 2 & 2 \\ 4 & 5 & 5 & 4 \\ 2 & 5 & 7 & 5 \\ 2 & 4 & 5 & 3 \end{bmatrix}$$

Let R be an example raster, and let L_c denote the number of equal-valued neighbors for each cell in R. The measure L_c considers the eight neighboring cells for interior cells, five neighbors for boundary cells, and three neighbors for corner cells. For this raster, the sum of L_c over all cells is 62, yielding $L = 62/16 = 3.88$.

We also investigate compressibility over the raster time series model. In this context, we apply a SFC to each raster in the time series and concatenate the resulting sequences. The goal is to determine the behavior of our approach for temporal locality and cyclicity in a raster time series. Cyclicity refers to a sequence of raster snapshots that repeats over time. Both cases produce similar rasters that generate repeated subsequences after linearization, which repetitiveness measures can leverage.

5.4.2 An alphabet-sensitive repetitiveness measure, δ_Δ

$$\delta_\Delta = \max \left\{ \frac{\sum_{s \in S_{k\times k}(M)} (\max(s) - \min(s))}{k^2} \right\} \tag{5.5}$$

We introduce δ_Δ, a 2D compressibility measure sensitive to the raster's alphabet. This measure is a δ_{2D} [31, 140] variant. The δ_{2D} measure seeks to maximize the ratio $\frac{S_{k\times k}(M)}{k^2}$ over a matrix M, where $S_{k\times k}$ represents the set of distinct $k \times k$ submatrices within the raster. The definition of δ_Δ is provided in Equation 5.5, where $\max(s)$ and

$\min(s)$ are the maximum and minimum values within submatrix s, respectively. The goal of δ_Δ is to maximize the difference between these maximum and minimum values, thus making the measure sensitive to the raster's specific values.

Consider the following raster example M:

$$M = \begin{bmatrix} 1 & 1 & 3 & 3 & 3 \\ 1 & 1 & 3 & 3 & 3 \\ 3 & 3 & 4 & 4 & 4 \\ 3 & 3 & 4 & 4 & 4 \\ 3 & 3 & 4 & 4 & 4 \end{bmatrix}$$

Let $S_{k\times k}(M)$ denote the set of distinct submatrices of size $k \times k$ within M, and let $d_k = |S_{k\times k}(M)|$ be their number. The δ_{2D} is a compressibility measure in the literature, defined as: $\delta_{2D} = \max_k \delta_{2D}(k)$, where $\delta_{2D}(k) = \frac{d_k}{k^2}$.

Our proposed measure δ_Δ accounts for value magnitudes inside each submatrix. We define $\delta_\Delta = \max_k \delta_\Delta(k)$, where $\delta_\Delta(k) = \frac{M_k}{k^2}$, and $M_k = \sum_{s\in S_{k\times k}(M)}(\max(s)-\min(s))$. Table 5.2 reports the values of $\delta_{2D}(k)$ and $\delta_\Delta(k)$ for all possible k values. We observe that $\delta_{2D} = 3$ when $k = 1$, and $\delta_\Delta = 2.3$ when $k = 2$. Both measures achieve their maximum values at different k, and the values also differ.

k	$\delta_{2D}(k)$	$\delta_\Delta(k)$
1	3	0
2	2	2.3
3	1	1.8
4	0.25	0.8
5	1/25	3/25

Table 5.2: Partial measures for $k = [1..5]$ on raster M

We apply different mapping $1{:}1$ of the alphabet permutation to reduce the δ_Δ measure. We evaluate all $3! = 6$ permutations $\pi : \{1, 3, 4\} \to \{0, 1, 2\}$, apply π to M, and recompute the measures. Table 5.3 presents the values of δ_Δ for all possible mappings. We observe that δ_Δ differs according to the mapping selected, with its minimal value being 1.5. On the other hand, δ_{2D} remains 2, independent of the mapping. This example illustrates the differences between δ_Δ and δ_{2D}.

Mapping	$\delta_\Delta(2)$	$\delta_\Delta(3)$	$\max(\delta_\Delta(k)) = \delta_\Delta$
(0,1,2)	1.5	1.3	1.5
(2,1,0)	1.5	1.3	1.5
(0,2,1)	2	1.3	2
(1,0,2)	2	1.8	2
(2,0,1)	2	1.3	2
(1,2,0)	2	1.8	2

Table 5.3: δ_Δ results for all possible mappings. $\delta_{2D} = 3$ for all cases.

5.5 Experimental framework

In this section, we present the experimental framework, including descriptions of the datasets and implementation details. Our source code and dataset are available at [117][3].

5.5.1 Datasets

Raster datasets We employed six different collections of rasters:

- `eua_join1X1` [72][4]: Consists of 25 rasters representing global climate data. Each cell has a spatial resolution of 1 km^2, showing the average temperature in Celsius with a precision of one decimal place. This collection comes from an original raster divided in 5×5 non-overlapping subrasters.

- `catalunya`[5]: Includes rasters representing a Digital Terrain Model (DTM) in Spain. Each cell in this collection has a spatial resolution of 5 m^2 and represents terrain altitude. This collection comes from an original raster divided into N^2 subrasters, where N ranges between 2 and 5. The original data use three decimal places, but we include in our experiments a variant with no decimal places. Therefore, this collection contains 8 subcollections called `cat_joinMxM.D`, where `D` is the number of decimal places and `MXM` is the relative raster size after the division of the original raster.

[3] `https://figshare.com/articles/dataset/Rasters_Datasets/27190659`
[4] `http://www.worldclim.org/tiles.php`
[5] `http://www.ign.es/`

- Miscellaneous RGB images: This collection was published by the University of Southern California[6]. From this source, we generated two collections: `misc`, that represents each image using a raster that represents the 3 RGB values for each cell in a 4-byte integer; and `misc_rgb`, that represents each RGB band using a different raster.

- `temporal`: Contains raster instants from the Rasters Time Series generated by the North American Land Data Assimilation System (NLDAS) [166]. NLDAS contains 11 temporal rasters, from which we obtained 26 uniformly distributed raster instants for each temporal raster.

- `APCP`: To study the impact of noise on compressibility measures, we generated a new semi-synthetic collection. This collection comes from a base raster representing accumulated precipitation in [mm] in the United States in 2018. The original raster contains 224 rows, 464 columns, and two decimal precision values. The raster contains 346 unique values in the range $[-1..344]$. The -1 value represents the cells that contain no-data values. These cells represent bodies of water in the raster. The rest of the cells contain valid data inside the continental part of USA.

  Using this base raster, we created a series of variants by altering values in randomly selected cells. Noise was introduced by considering three factors: the area of modification, the number of cells to modify, and the maximum difference between the original and new values in each cell. For the first factor, we consider two areas: *complete* (the entire raster) and *inner* (the area with valid values). For the second factor, we defined the *percentage of changed cells* (PCC) as the ratio c/N, where c is the number of modified cells, and $N = 103,936$ is the total number of cells in the raster. For the third factor, after selecting the cells to modify, we add to the original value a random value between 1 and a maximum threshold u, where $u = R \times PV$, with $R = 346$ representing the range of values in the base raster, and PV is the *percentage of variation*.

- Synthetic Collections: We generated two synthetic raster collections to study the sensibility to alphabet. The first collection, *version 1*, starts with a base raster of

[6] `https://sipi.usc.edu/database/database.php?volume=misc`

512 $\times$ 512 with an alphabet $\Sigma = \{100, 200, 300, 400, 500\}$. These values are uniformly randomly distributed. We generate different variants by scaling all values in the raster by a multiplicative growth factor α. The second collection, *version 2*, comprises 100 rasters with similar properties to the base raster in the first collection: rasters of size 512 $\times$ 512, with the same alphabet and random distribution. The difference lies in the frequency of values, skewed according to the frequency set $f = \{0.5, 0.25, 0.125, 0.0625, 0.0625\}$. Value-frequency pairs within Σ and f are randomly assigned.

Table 5.4 outlines the properties of the real-world datasets collections.

Collection name	#rasters	rows $\times$ cols	min value	max value	#unique values
eua_join1X1	25	3,600 $\times$ 3,600	0	542	252
cat_join1X1.3	25	4,100 $\times$ 5,849	0	2,247,887	779,405
cat_join1X1.0	25	4,100 $\times$ 5,849	0	2,247	868
cat_join2X2.3	16	8,242 $\times$ 11,737	0	2,247,887	1,066,043
cat_join2X2.0	16	8,242 $\times$ 11,737	0	2,247	1,201
cat_join3X3.3	9	12,403 $\times$ 17,643	0	2,247,887	1,304,704
cat_join3X3.0	9	12,403 $\times$ 17,643	0	2,247	1,494
cat_join4X4.3	1	16,484 $\times$ 23,524	0	2,247,887	1,829,334
cat_join4X4.0	4	16,564 $\times$ 23,564	0	2,247	1,761
misc	39	459 $\times$ 459	2,147,483,647	4,293,874,284	29,323
misc_rgb	117	459 $\times$ 459	8	242	183
temporal	297	224 $\times$ 464	-2,163	10,425,597	6,597

Table 5.4: Main statistics of the used real-world dataset collections

Temporal raster datasets For the study of compressibility measures on temporal rasters, we used a collection containing four raster time series provided by the North American Land Data Assimilation System (NLDAS) [166]. This collection incorporates various raster time series representing natural phenomena recorded across the United States in 2018. Each temporal raster has a spatial resolution of 1/8 degrees, a temporal resolution of one hour, and a precision of two decimal places. Each temporal raster contains 224 rows, 464 columns, and 2664 instants. This collection contains 11

datasets, but in this work we present the results of 5 representative datasets: `DSWRF`, `DLWRF`, `PEVAP`, `CONVfrac`, and `SPFH`.

Additionally, we generated a collection of temporal rasters to analyze the impact of repetitiveness measures on cyclic temporal rasters. This collection is called *cycle*. Each dataset contains 64 rows, 64 columns, and 64 instants, with values ranging from 1 to 100. The collection includes six datasets, denoted as `c-x`, where x represents the cycle length. Each dataset contains x instants that repeat consecutively until the total number of instants. The possible values for x are in the set $X = \{2, 4, 8, 16, 32, 64\}$.

5.5.2 Implementation

Our implementation utilized C++ and Python. C++ code was compiled with GCC version 11.4.1 with the following flags: `-O9 -g3 -std=c++17`. Additionally, we use Python version 3.9.18. The principal external libraries employed were the *Succinct Data Structure Library* (SDSL) [68][7] and *Exploratory Spatial Data Analysis* (ESDA)[8]. The z measure implementation was adapted from Goto and Banai's code [70][9]. The v measure implementation follows Nicola Prezza's approach [123][10]. The RePair heuristic, applied to compute the g measure, was implemented using an enhanced RePair [100] version based on Gonzalo Navarro's implementation[11]. Regarding compressors, we used version 1.0.8 of bzip, version 1.12 of gzip, and version 4.8.1 of NetCDF, setting each to maximum compression. Finally, Fernando Silva's k^2-raster implementation[12] version 3.1 was used.

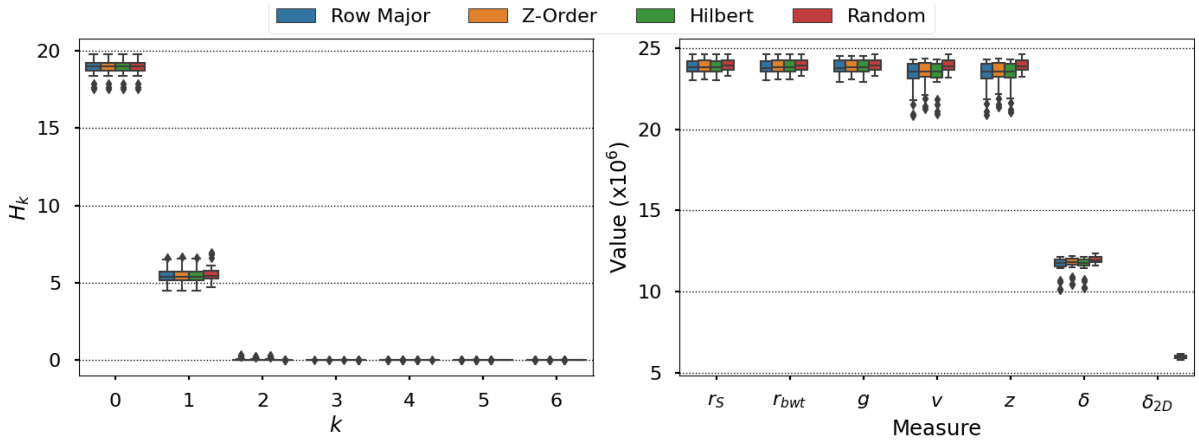
5.6 Experimental results

5.6.1 Comparison of one-dimensional compressibility measures across SFCs

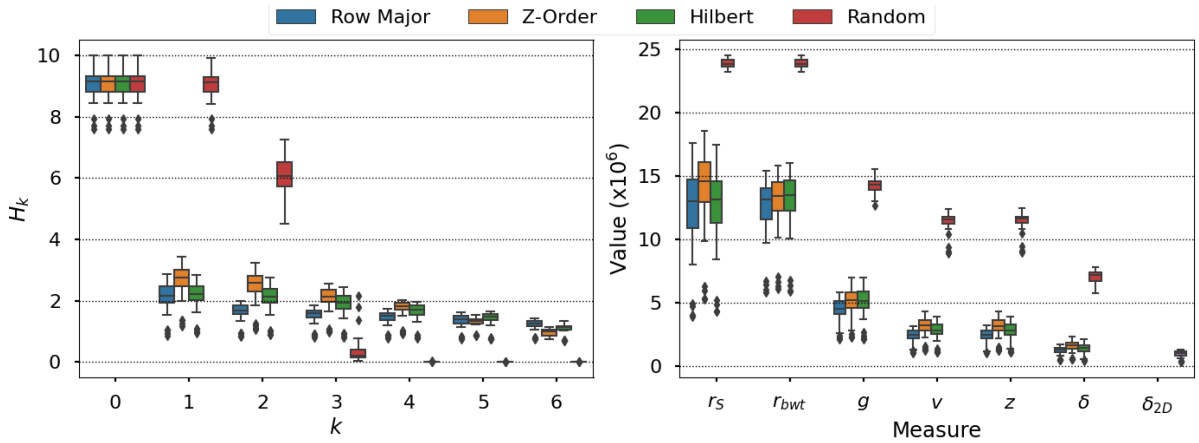
This section presents the impact of the different Space-Filling Curves (SFCs) (see Section 5.4.1) on the compressibility measures. To provide context for the results, we

[7] `https://github.com/simongog/sdsl-lite`
[8] `https://pysal.org/esda/`
[9] `https://code.google.com/archive/p/lzbg/source/default/source`
[10] `https://github.com/nicolaprezza/lex-factorization`
[11] `https://www.dcc.uchile.cl/gnavarro/software/repair.tgz`
[12] `https://gitlab.lbd.org.es/fsilva/k2-raster`

include a fictitious SFC called *Random curve*, which consists of a random permutation of the raster cells, simulating a linearization without spatial locality. We present the results from the `cat_join1X1.3` and `cat_join1X1.0` collections as representative results, since the other datasets behave similarly.



(a) `cat_join1X1.3` Collection (L measure mean: 0.04, std: 0.04)



(b) `cat_join1X1.0` Collection (L measure mean: 3.67, std: 1.27)

Figure 5.7: Compressibility measures on `cat_join1X1` Collection. H_k is shown at the left, and the remaining measures are shown at the right

Figure 5.7a shows the result values for the combination of each SFC and compressibility measure over `cat_join1X1.3` collection (values with 3 decimal places), while Figure 5.7b shows the results over `cat_join1X1.0` collection (values without decimal places). For `cat_join1X1.3` collection, the results indicate that the four curves exhibit a similar behavior. Measures r_S, r_{bwt}, g, v, and z show similar values between them. For `cat_join1X1.0` collection, row-major, Z-Order, and Hilbert curve perform similarly,

while random curve produces higher values than the other three curves for compressibility measures (except for H_0 and H_k for $k \geq 3$).

The similar behavior between the SFCs indicates that row-major achieves a similar level of locality preservation to the other two SFCs. The convergence of the three SFCs with the Random curve on `cat_join1X1.3` collection is related to the high level of precision of its values. When precision increases, the "noise" in the raster also increases, hence enlarging the alphabet and reducing spatial locality, which challenges raster compression. The sequences generated by the three SFCs exhibit a low level of repetitiveness and compressibility (similar to the case of Random curve) and thus the 1D measures cannot take advantage of such property (i.e., the values that each measure obtains in `cat_join1X1.3` are larger than those obtained in `cat_join1X1.0`).

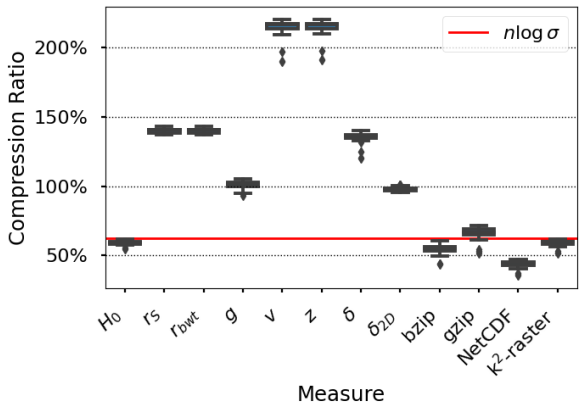
The results confirm the relation between the measures documented in the literature [122]. In Figure 5.7b, we have $r_S \approx r_{bwt} \approx 10\delta_{2D}$, $g \approx 4\delta_{2D}$, $z \approx v \approx 2\delta_{2D}$ and $\delta \approx 1.2\delta_{2D}$. In Figure 5.7a we have $r_S \approx r_{bwt} \approx g \approx z \approx v \approx 2\delta \approx 4\delta_{2D}$. The relation $r_S \approx r_{bwt}$ arises due to the nature of both measures, which count the number of runs within a sequence. The relation $\delta > \delta_{2D}$ demonstrates the effectiveness of 2D measures in capturing the repetitiveness in a raster, compared to the application of 1D measures. Moreover, both δ and δ_{2D} continue being lower than the other measures, owing to the nature of these measures.

Concerning the H_k behavior, these results are related to the number of contexts or the amount of different subsequences of size k. When increasing k, the number of contexts increases until it reaches all possible subsequences of size k. This behavior decreases the frequency of each context to 1 and decreases the H_k value until it approaches 0. These results imply that an empirical entropy-based compressor can reduce the sequence size represented by increasing the value k; however, the implicit dictionary of contexts of length k grows significantly.

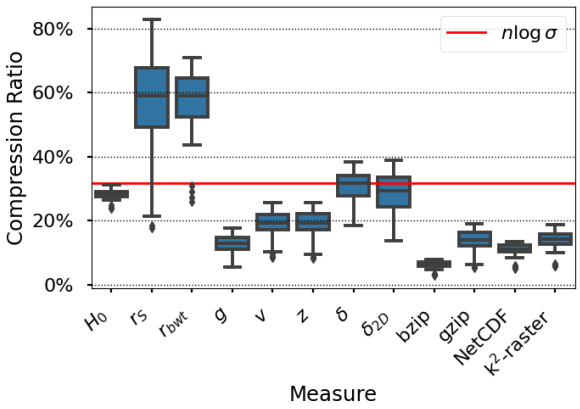
5.6.2 Estimation of raster compressibility based on compressibility measures

In this section, we examine the compressibility of a raster based on different techniques derived from compressibility measures. We define the compression ratio as the quotient of the compressed raster size and the original size, assuming a 4-byte integer

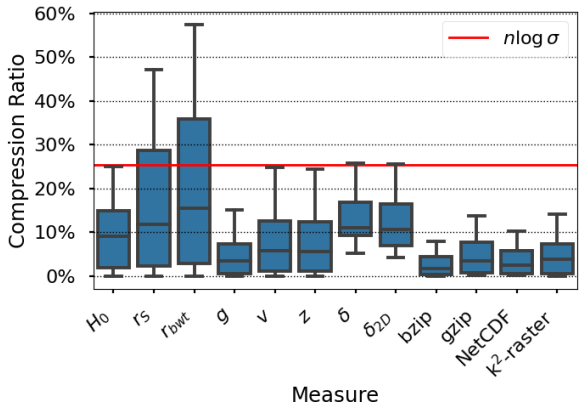
representation per cell. To compute the raster compressed size, we estimate the minimum number of bits required to explicitly represent each phrase, rule, or subsequence each measure uses in a method that allows for the recovery of the original sequence. To complement our study, we include the compression ratio of compressors and k^2-raster described in Section 5.2. Additionally, we include a comparison with a naive fix-length encoder using $n\lceil \log_2 \sigma \rceil$ bits. For 1D measures, we use the row-major curve as the linearization method.



(a) `cat_join1X1.3` collection (L measure mean: 0.04, std: 0.04)



(b) `cat_join1X1.0` collection (L measure mean: 3.67, std: 1.27)



(c) `eua_join1X1` collection (L measure mean: 6.61, std: 1.28)

Figure 5.8: Raster compression ratios for `cat_join1X1` and `eua_join1X1` collections

Figure 5.8 presents the results for three collections: `cat_join1X1.3` (Figure 5.8a), `cat_join1X1.0` (Figure 5.8b), and `eua_join1X1` (Figure 5.8c).

The scenario in Figure 5.8a is an extreme case where compression is challenging, as reflected in the performance of each measure. Only the measure H_0 may achieve better compression than fixed-length encoding. All compressors and the k^2-raster achieve a compression ratio below 100%, but similar to the naive encoder. The case of Figure 5.8b is different because the ratios reported by all measures are lower than 100%, showing the possibility of achieving compression. Measures δ, δ_{2D}, gzip, and k^2-raster reach an average compression below $n\lceil \log_2 \sigma \rceil$. In the collection shown in Figure 5.8c all measures reach better ratios than fixed-length encoding, where only r_S and r_{bwt} require more than $n\lceil \log_2 \sigma \rceil$ in a few cases (the average is always below such value). The last scenario contains the lowest number of unique values, reducing the measure values and compressor sizes.

We also observed a relation between measure L (reported in the caption of the figures) and compression ratio. Lower L values makes compression harder, as evidenced by the behavior of the compression ratios. This behavior is also related to alphabet size, where decreased spatial locality correlates with a larger alphabet.

5.6.3 Correlation between measures

This section shows the *Spearman correlation* [5] between the compressibility measures, compressor sizes, k^2-raster size, spatial locality, and spatial autocorrelation, presented in Section 5.2. To avoid spurious correlations due to raster size, some measures (r_S, r_{bwt}, z, v, g, δ, δ_{2D}, compressor sizes, and k^2-raster size) were normalized by dividing their values by the original raster size, as the number of cells that contain. For 1D measures, we used the row-major curve as the previous linearization. We apply this analysis to seven different collections of rasters: `cat3` (includes the rasters from `catalunya` with three decimal places), `cat0` (includes the rasters from `catalunya` with zero decimal places) `cat` (includes the rasters from `catalunya` with 3 and 0 decimal places), `misc`, `misc_rgb`, `temporal` and the union of all the mentioned collections (named `all`).

Figure 5.9 shows the Spearman correlation results for each collection. We observe: *i)* A strong positive correlation between compressibility measures and the sizes of compressors and the CDS, *ii)* A consistent negative correlation of the measure L

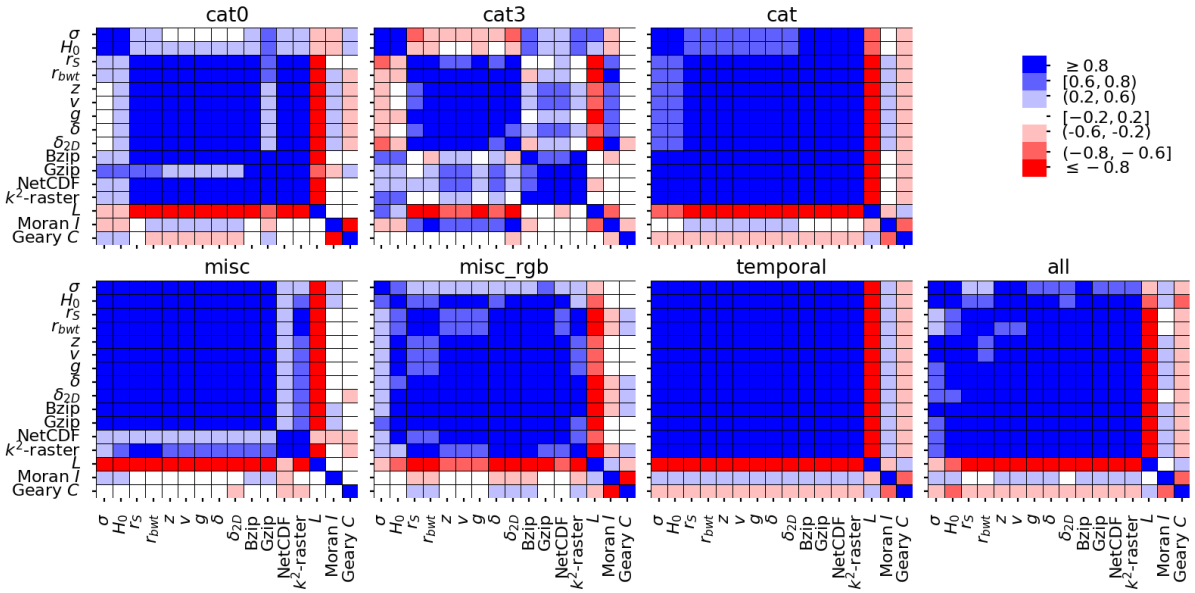


Figure 5.9: Spearman Correlation on measures for each collection

with most other measures, *iii)* Near-zero correlation values between spatial autocorrelation measures σ, H_0, Moran's I, and Geary's C, indicating no significant correlation, *iv)* Focusing on the compressibility measures group (r_S, r_{bwt}, g, v, z, δ, and δ_{2D}), the Spearman coefficient never falls below 0.6, highlighting a significant mutual correlation.

Figure 5.10 summarizes all the collections where we computed the Spearman correlation. For each collection, we grouped the pairs that presented a high correlation (> 0.8) and the pairs that presented a low correlation (< -0.8). Figure 5.10 counts the collections with a high or low correlation and shows them in shades of blue or red, respectively. Among these: *i)* δ_{2D} appears most frequently, correlating strongly with five measures (r_S, r_{bwt}, g, v, and z), *ii)* r_S appears in the fewest highly correlated pairs (only with r_{bwt} and δ_{2D}), *iii)* the measures z, v, and g consistently shows the highest correlations across collections. The similarity in definitions of z and v explains their strong correlation, while their link with g is an interesting new finding.

When observing the correlation behavior between the compressors and the k^2-raster, Figure 5.10 shows four pairs show a correlation greater than 0.8 in six of the seven collections: bzip with gzip, NetCDF with a k^2-raster, NetCDF with gzip, and bzip with a k^2-raster. Only in the `misc` collection do NetCDF-bzip and NetCDF-gzip show correlations below 0.6 (see Figure 5.9).

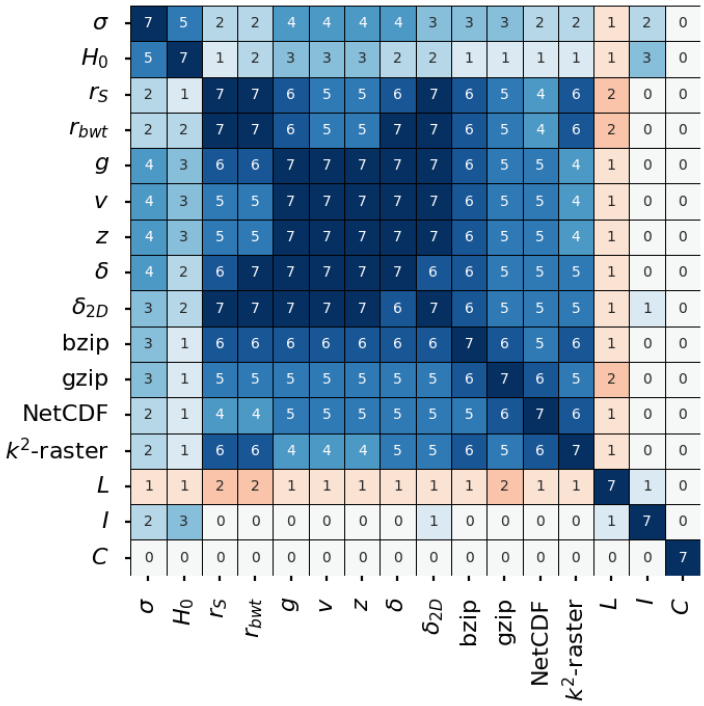


Figure 5.10: Spearman Correlation resume

When comparing the correlation between compressibility measures and the size of the compressors sizes and the k^2-raster size, the Spearman coefficient reaches a value of at least 0.2 for all collections except `cat3`. In Figure 5.10, several pairs with a correlation greater than 0.8 arise in six of the seven collections. These pairs occur in bzip with all measures and in the k^2-raster with r_S and r_{bwt}. Observing the behavior of the L measure, we notice that the Spearman coefficient reaches a maximum value of -0.2 in all collections except `cat3`. For this collection, correlation values are close to 0, and just a few cases have positive correlations. In `cat3`, it is difficult to determine any correlation due to its distribution of points (see Figures 5.11, 5.12 and 5.13 for more details).

Figures 5.11, 5.12 and 5.13 present the principal relations between all measures considered: *i)* Figure 5.11 shows relations between compressibility measures (r_S, r_{bwt}, z, v, g, δ, and δ_{2D}) and L, *ii)* Figure 5.12 shows relations among compressors and CDS size, and *iii)* Figure 5.13 shows relations between measures and compressors with CDS. The `cat3` and `misc` collections exhibit similar behavior, likely due to their

large alphabet sizes. The temporal and misc collections display a forked line in several graphics. In the case of misc, this is due to the difference between the color and gray-scale images. In the case of temporal, the rasters of the PEVAP dataset are separated from the rest of the points. On the other hand, the temporal collection displays the most defined lines in all graphics. This behavior also appears in Figure 5.9, where the positive and negative correlations are well-defined. From Figures 5.11 and 5.13, we can observe a negative correlation between L (spatial locality) and the compressibility measures, compressors, and k^2-raster size. The strongest negative correlations are with r_S and r_{bwt}. This outcome suggests that higher spatial locality corresponds to lower compressibility, smaller compressor, and k^2-raster sizes. In Figure 5.13, a positive linear correlation exists between compressibility measures and the size of the compressors and the k^2-raster, whereas L shows a negative linear correlation with these sizes. These tendencies are mainly formed with the collections temporal, cat0 and misc_rgb. Regarding value ranges, weighted compressibility measures range from 0 to 1 (see Figures 5.11 and 5.12). In the worst case (all cells are distinct), these measures will equal the size of the alphabet, and its weight will be 1. The same applies to the compressors and CDS sizes, except they range from 0 to 4 (see Figures 5.12 and 5.13). In the worst case, the compressor must represent each raster value in 4 bytes. Of the three compressors and the CDS, bzip has the lowest range, reaching only 3. This result suggests that this compressor is the most efficient.

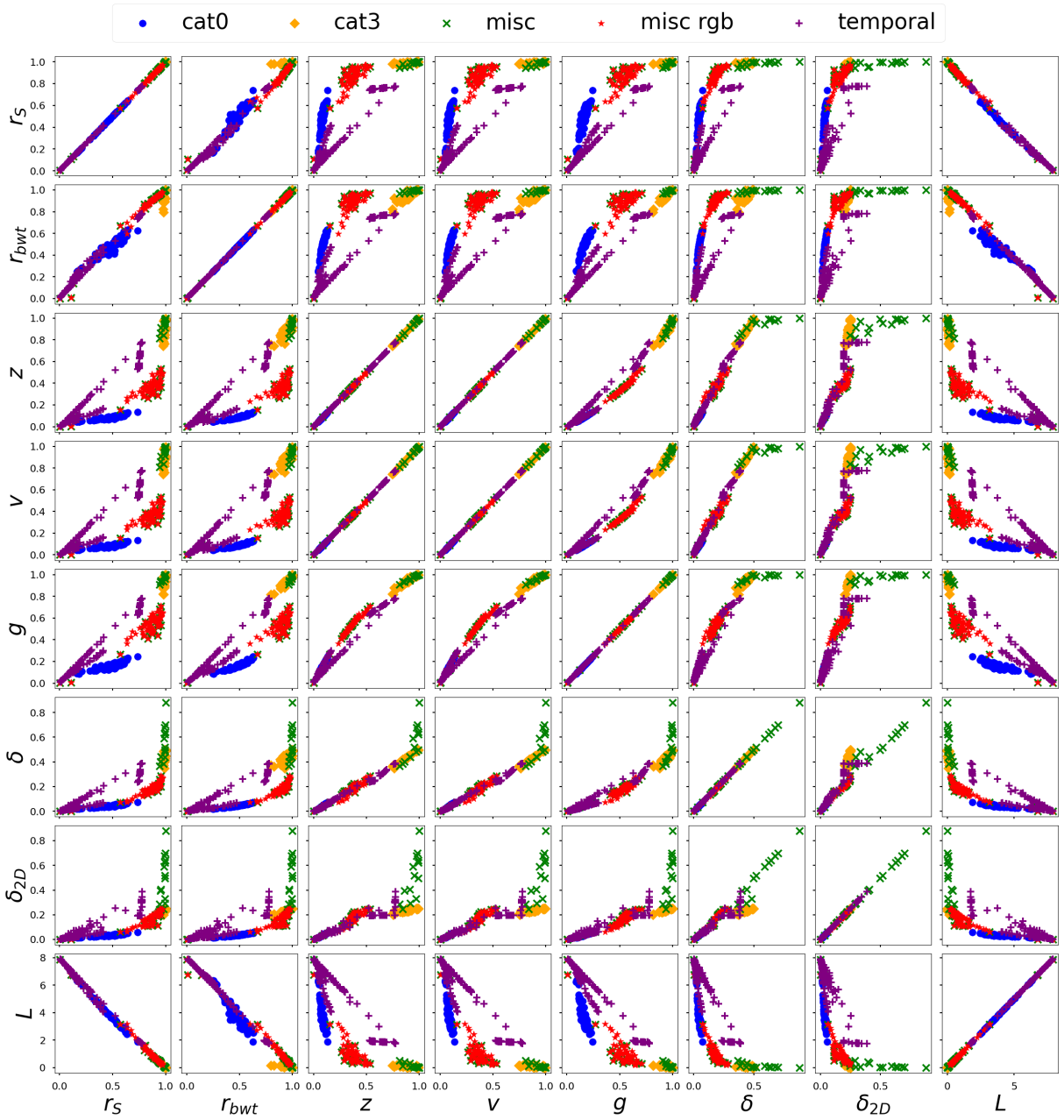


Figure 5.11: Relation between measures

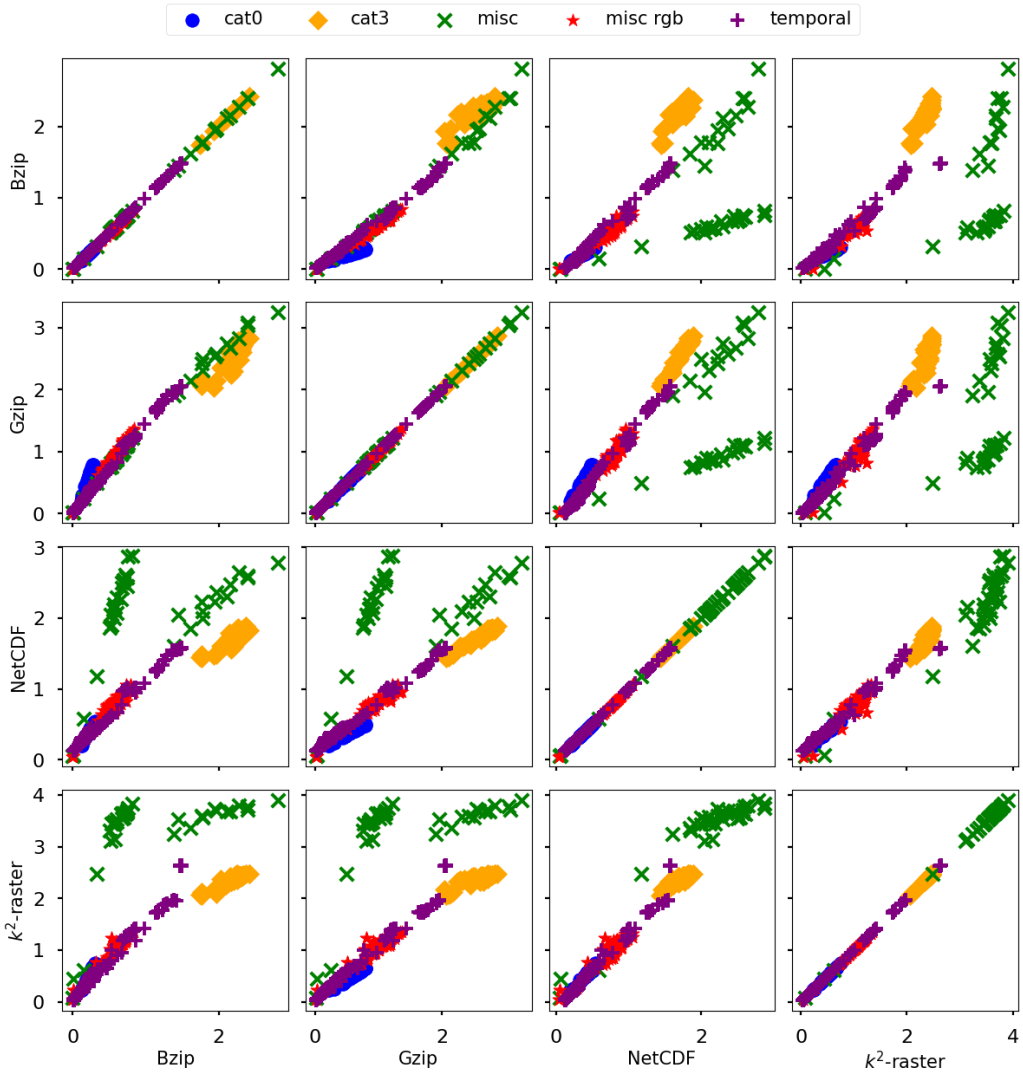


Figure 5.12: Relation between compressors

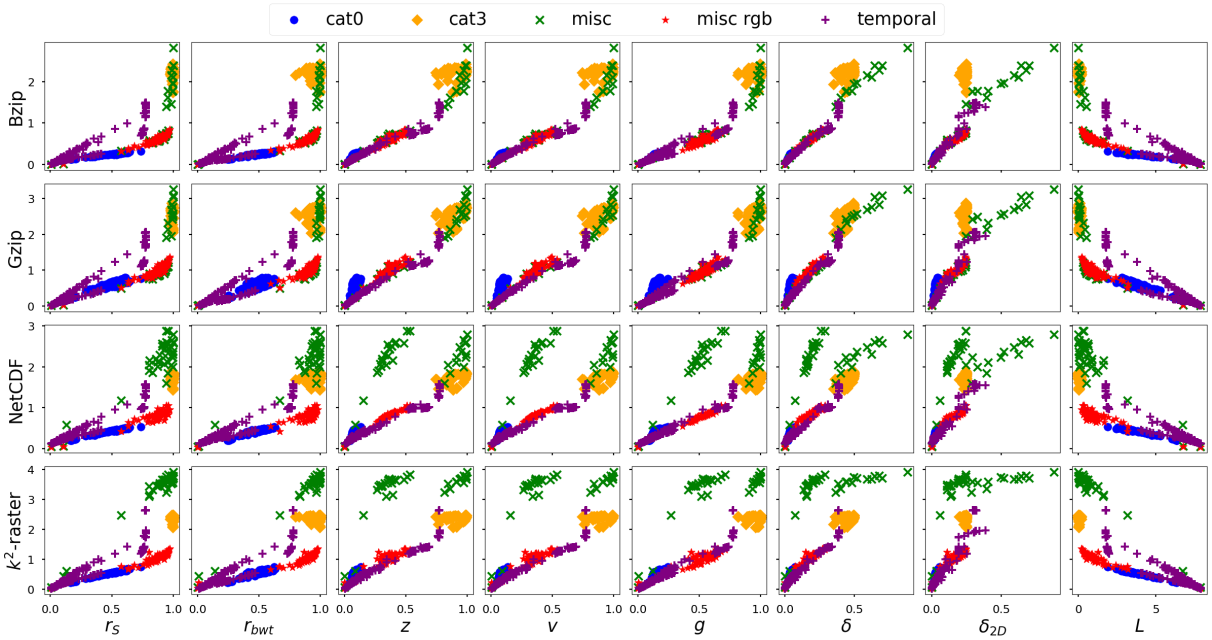


Figure 5.13: Relation between compressors and measures

5.6.4 Sensitivity to noise

This section examines the impact of noise on compressibility measures and the sizes of compressors and the k^2-raster. In a raster, "noise" refers to irregular or unwanted spatial variations in pixel values. To study this property, we constructed a semi-synthetic dataset, APCP, in which we considered three factors: the predefined region where changes are applied (complete or inner), the percentage of changed cells (PCC), and the maximum difference between the original value and the new replacement value (PV). The percentage of sensitivity to noise (PSN) is the percentage for each measure is defined as the ratio between the measure's value on the modified raster and its value on the original raster. For this analysis, we employed `APCP` collection presented in Section 5.5.

Figure 5.14 displays PSN values for each measure with PCC = 50% and PV = 50%, with the predefined region as a variable. Figure 5.15 shows PSN values on r_S measure. Each measure considered except L, Moran's I and Geary's C has a similar behavior. We highlight the following results: *i)* All three factors influence noise intensity. PCC has a more significant impact on both regions than PV. *ii)* The complete region

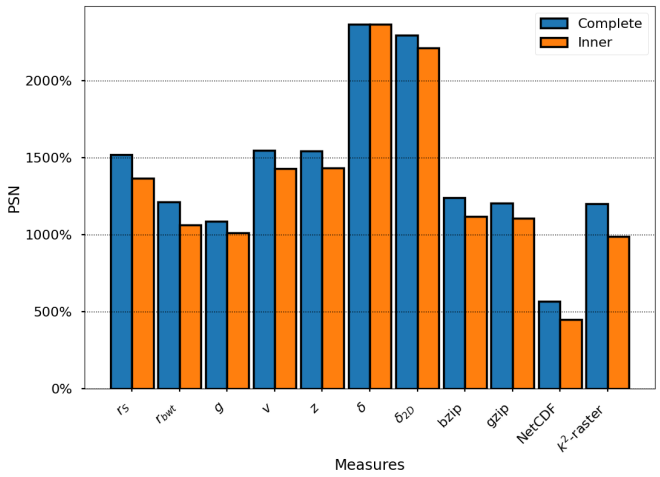
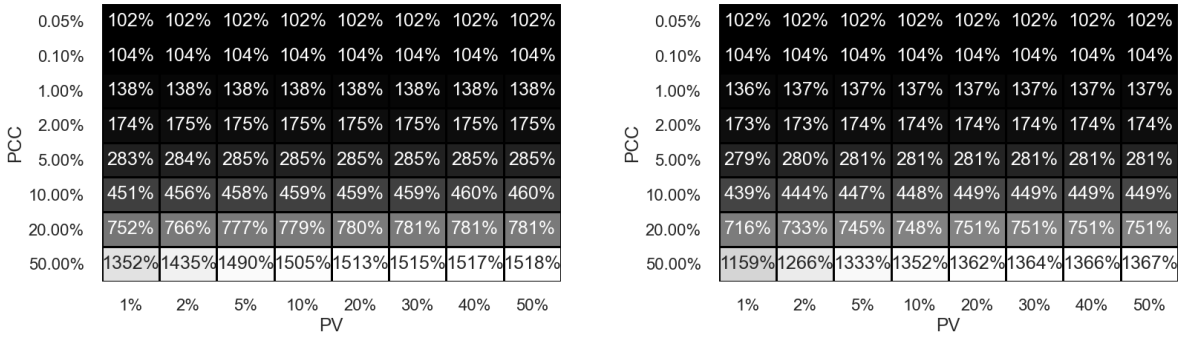


Figure 5.14: PSN for each measure (PCC: 50%, PV 50%)



(a) Row major + r_S, `complete` region

(b) Row major + r_S, `inner` region

Figure 5.15: PSN values on compressibility measures

shows a higher PSN value than the inner region, which is observable in both cases with PCC = 50% and PV = 50%. *iii)* The PSN value in the complete region is always equal to or higher than that in the inner region. *iv)* The highest PSN values are observed in measures δ and δ_{2D}, while the lowest arise with the NetCDF compressor size (see Figure 5.14).

5.6.5 Analysis of one-dimensional compressibility measures on temporal rasters

This section presents the experimental results analyzing the relation between 1D compressibility measures, compressors, and Compact Data Structures (CDS) applied to raster time series. The goal is to assess the influence of temporal locality and cyclicity

on compression performance (see Subsection 5.4.1).

We evaluate two linearization approaches based on the row-major curve: *Plain*, which linearizes each instant independently and concatenates the results, and *Depth*, which traverses the grid once and collects values across instants for each cell. Plain linearization emphasizes the spatial traversal of the raster, whereas Depth prioritizes a temporal traversal. The datasets used in this study include the `NLDAS` collection, which represents real-world scenarios, and the *cycle* synthetic collection, designed to assess the impact of instant cyclicity effects (see Section 5.5).

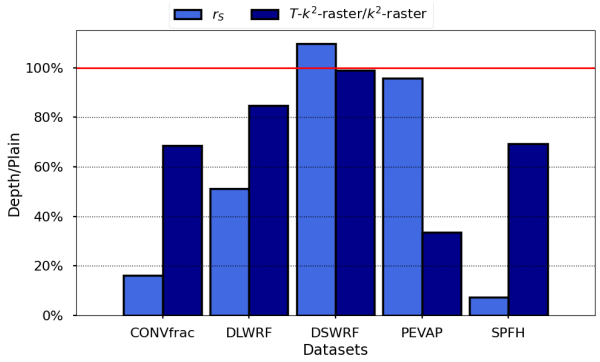
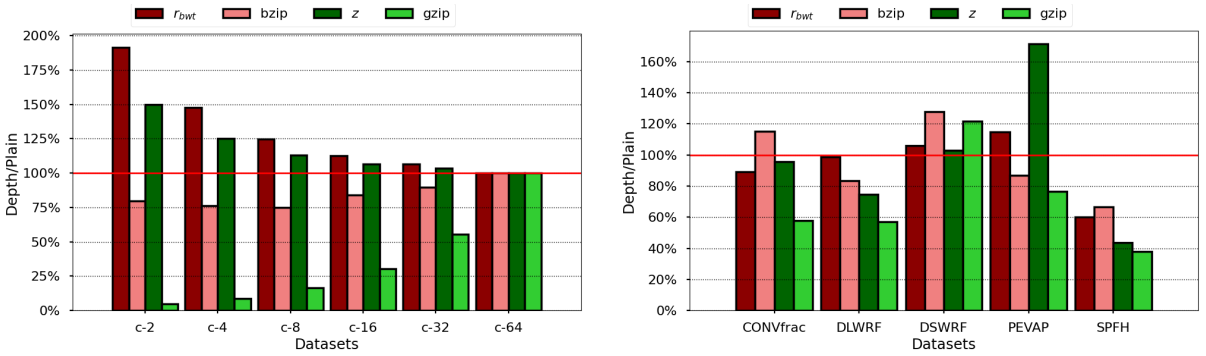


Figure 5.16: Depth/Plain ratios comparison for temporal locality. Red line at 100% indicates that both linearizations approaches generate the same results.

We begin by analyzing temporal locality using the r_S measure, which is insensitive to repetitiveness and suitable for isolating spatial and temporal patterns. The analysis begins by computing the r_S ratio Depth/Plain linearizations. As shown in Figure 5.16, in most cases, this ratio is below 100%, suggesting that temporal locality (captured by Depth) is stronger than spatial locality (captured by Plain). A notable exception is observed in the `DSWRF` dataset.

To further investigate the effect of temporal locality on compression, we compare two CDSs: *i)* indexing each instant as a separate k^2_H-raster, forming a Collection of k^2_H-rasters, and *ii)* indexing the entire temporal raster as a T_H-k^2-raster. The collection of k^2_H-rasters is insensitive to temporal locality, whereas the T_H-k^2-raster exploits temporal locality to improve compression. The results, also shown in Figure 5.16, confirm that the T_H-k^2-raster achieves superior compression, with all ratios falling below 100%.



(a) `Synthetic` collection

(b) `NLDAS` collection

Figure 5.17: Depth/Plain ratios comparison for repetitiveness measures. Red line at 100% indicates that both linearizations approaches generate the same results.

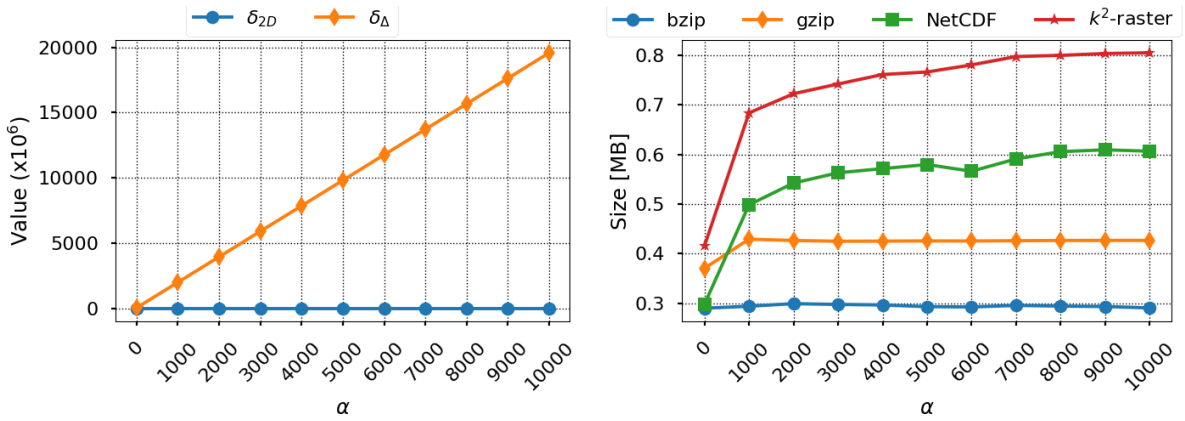
Next, we assess the role of repetitiveness in compression performance. We consider four measures: r_{bwt}, bzip, z, and gzip. On the one hand, bzip applies the Burrows-Wheeler Transform (BWT) in its compression process; for the other, r_{bwt} requires the BWT computation to obtain its value result. Likewise, gzip applies Lempel-Ziv to its compression process, and z requires Lempel-Ziv for its computation. For each case, we compute the ratio of the measure between Depth and Plain linearizations. Figure 5.17 summarizes these results, while Figures 5.17a and 5.17b present the outcomes for the `cycle` and `NLDAS` collections, respectively.

In the synthetic `cycle` dataset, we can observe a significant difference between the ratios of the measures and their corresponding compressors for small cycle lengths. These differences decrease as the cycle length increases, converging toward an equilibrium point at 100%. In the real-world `NLDAS` datasets, the results reveal a heterogeneous behavior across datasets. For example, in `PEVAP`, the difference between the ratios is considerable, whereas in `DLWRF`, the difference is minimal. In `CONVfrac` and `DSWRF`, the ratio for bzip exceeds that of r_{bwt}, while in `DSWRF`, the ratio for gzip is higher than that of z.

These inconsistencies highlight an important insight: although both bzip and gzip are conceptually tied to specific measures (BWT and Lempel-Ziv, respectively), additional stages in the compression pipeline influence their actual compression performance. As a result, the empirical behavior of the compressors may deviate from the theoretical expectations based solely on their underlying measure.

5.6.6 Evaluation of δ_Δ

This section evaluates the δ_Δ measure described in Section 5.4.2. We evaluated the measure behavior based on different aspects presented in the raster and compared this with δ_{2D} and the structures' behavior.



(a) δ_{2D} and δ_Δ vs multiplicative growth factor α ($\hat{k} = 2$)

(b) compression size vs multiplicative growth factor α

Figure 5.18: Comparison between δ_{2D}, δ_Δ, compressors and k^2-raster on *version 1* collection

We first analyze results using the synthetic raster collection *Version 1* (see Section 5.5). Figure 5.18 illustrates the behavior of the 2D measures, compressors sizes, and the k^2-raster concerning the multiplicative grown factor α, that scales all values in the base raster of 512 $\times$ 512 cells. Figure 5.18a shows that δ_{2D} remains constant while δ_Δ exhibits a clear linear growth. Let $\hat{k}$ be the k value where δ_{2D} and δ_Δ reach their respective maximum. We obtain the same constant value $\hat{k} = 2$ for both measures. The similar value for $\hat{k}$ demonstrates that the principal influence is the values instead of the submatrix size. Figure 5.18b shows that both NetCDF and the k^2-raster are influenced by the multiplicative grown factor α.

The results indicate that the δ_Δ measure reaches its minimum value when the alphabet values have a minimum difference, which is consistent with its definition. The alphabet that minimizes δ_Δ contains fully contiguous values. Furthermore, we verify that δ_{2D} is insensitive to variations in the magnitude of the alphabet values, marking a clear distinction from δ_Δ. Finally, we observe that variations in the value magnitude also

affect NetCDF and k^2-raster. In particular, the k^2-raster compaction technique relies on storing the differences between raster values, which explains the impact of value magnitude variations on its performance. These findings also highlight the usefulness of δ_Δ in estimating the compressibility of a raster.

Next, we evaluate the influence of values distributions along a raster in δ_Δ measure. For this, we used the synthetic raster collection *Version 2*, described in Section 5.5. Our results highlight that δ_Δ is sensitive to the distribution of the values, in particular to the frequency of values.

δ_Δ measure is sensitive to changes in raster values and their frequencies. It is well-suited to estimating the compression level of a raster under compressors that are similarly influenced by raster values, as is the case with NetCDF and k^2-raster. The linear increase in δ_Δ demonstrates that the measure's value rises as the range of raster values expands. The δ_Δ value increases as the alphabet's values become more dispersed. Extreme values exert the most significant influence on the measure; an increase in frequency corresponds to an increase in the δ_Δ value.

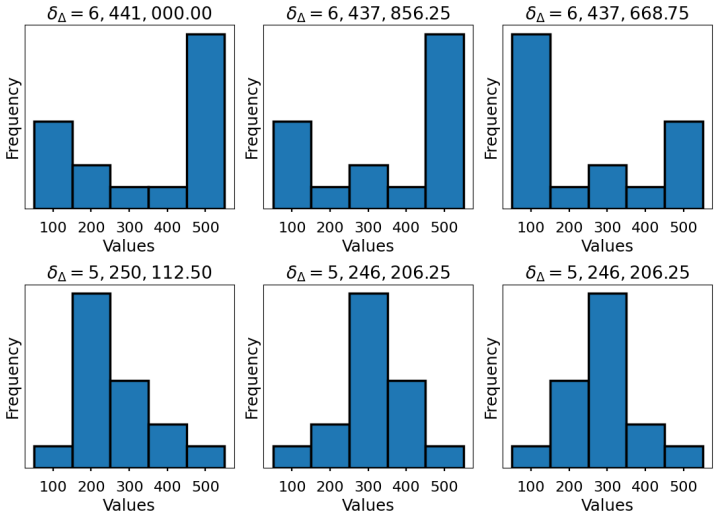


Figure 5.19: δ_Δ results for synthetic datasets on *version 2* collection ($\hat{k} = 4$, $\delta_{2D} = 16,144$)

Figure 5.19 shows that δ_Δ is sensitive to frequency distribution: rasters dominated by extreme values yield higher measure values, whereas distributions favoring central values produce lower values. Additionally, δ_{2D} remains constant with a $\hat{k} = 4$ for all

cases.

5.7 Conclusions and future work

In this chapter, we estimate raster compressibility by introducing two different approaches. First, we propose the combination of a Space-Filling Curve (SFC) with a one-dimensional compressibility measure. Second, we introduce $\delta_{\triangle}$, a novel two-dimensional compressibility measure that is sensitive to the raster alphabet. In the experimental section, we examine how these approaches influence or relate to data compression, spatial locality, and the impact of noise. Finally, we analyze a possible generalization to raster time series.

After developing our study, we highlight the following findings:

- The three SFCs studied, row-major, Hilbert and Z-order, exhibit similar behavior in terms of compressibility. For datasets with low spatial locality, the three SFCs behave similar to a random SFC.

- We empirically confirmed the relationship $r_S \approx r_{bwt} \geq g \geq v \approx z \geq \delta > \delta_{2D}$. The measures r_S, r_{bwt}, g, z, and v can reach similar values when the raster alphabet is large (i.e. when almost every cell has a different value).

- The behavior of the empirical entropy of order k and the number of contexts of size k shows that an empirical entropy-based compressor can reduce the sequence size represented by increasing the value k; however, the implicit dictionary of contexts of length k grows significantly.

- The correlation analysis shows a positive correlation between the compressibility measures and the size of the compressors, including the k^2-raster, and a negative correlation between these measures and the spatial locality measure L. Overall these correlations, the following ones stand out for its consistent correlation among different datasets: the trio of measures z, v, and g; bzip with gzip, the k^2-raster, and the compressibility measures; the k^2-raster with NetCDF, r_S, and r_{bwt}; and NetCDF with gzip.

- The influence of noise in a raster can significantly affect compressibility measures' value, the compressors' size, and the k^2-raster. We observe that the factor with the most significant impact is the number of cells affected by noise, that is, those affected by replacing a random value different from the original.

- When analyzing the compression behavior of raster time series, we find that the r_S measure captures temporal locality effectively when applied with cross-instant linearization. Similarly, the T_H-k^2-raster outperforms k^2-raster Collection. In contrast, repetitiveness measures and related compressors show irregular behavior across datasets.

- When analyzing compression behavior using the cyclicity of rasters in a raster time series, we can observe disparate results when comparing r_{bwt} with bzip, and z with gzip without establishing a definitive trend.

- The $\delta_\triangle$ measure is influenced by two important factors: the raster alphabet and the frequency distribution of its symbols. The measure increases as the alphabet size and the skewness of the frequency distribution grow. $\delta_\triangle$ is a good predictor of compressors with similar behavior, such as NetCDF and the k^2-raster.

As future work, we believe that a theoretical approach to the $\delta_\triangle$ measure could help define other relevant properties. Additionally, it would be interesting to explore the possibility of designing compressors or Compact Data Structures (CDS) inspired by this measure. Finally, regarding raster time series, our work is preliminary and paves the way for a more in-depth study. In particular, the SFCs we employed prioritize either spatial or temporal locality, but there are alternatives such as the Hilbert 3D curve [77], which can balance both properties. It would be interesting to investigate whether such a balance can reduce compressibility measure values in certain datasets. This also opens the door to new compressors and CDS for raster time series, an emerging line of research.

Chapter 6

Massive Parallel Construction of a k^2-raster

6.1 Introduction

This chapter presents a parallel algorithm for constructing the Compact Data Structure (CDS) k^2-raster [97, 98]. Although this structure provides efficient storage and query capabilities, its construction is computationally demanding compared to the queries it supports. The sequential `bottom-up` approach requires $O(nm)$ time for a raster of n rows and m columns [97, 98], making it expensive for large datasets.

Massively parallel computing provides a natural way to accelerate this construction. In this model, a set of interconnected processors executes multiple tasks (threads) simultaneously. The design of parallel algorithms involves key considerations such as data partitioning, synchronization, and interprocess communication. Unlike traditional parallelism limited by the number of processors, massively parallel computing can manage thousands of concurrent threads, enabling fine-grained parallelization [66]. General-purpose GPUs (GPGPUs) are the most common platform for this model.

The proposal parallel algorithm follows a bottom-up approach, constructing each node of the k^2-raster at the same level in parallel and allowing the pruning of the necessary nodes at the same level, also in parallel. The cells inside a subraster within the structure are spatially clustered, and subrasters do not overlap within the same level, facilitating data partitioning among threads. This design reduces the construction time to a complexity proportional to the tree height. Our proposal builds the arrays `Lmax` and `Lmin`, the values `rMax` and `rMin`, and the bitmap `T`, using CUDA[1] [129] on an Nvidia GPU. Experimental results demonstrate a significant reduction in construction time compared with the sequential implementation.

This chapter is organized as follows. Section 6.2 describes the related work about parallel computing applications on different CDSs. Section 6.3 describes our proposal.

[1] `https://docs.nvidia.com`

Section 6.4 reports the experimental process and results. Finally, Section 6.5 presents the principal conclusions and future work.

6.2 Related Work

Different algorithmic proposals related to Compact Data Structure (CDS) involve parallelism. Parallel computing was applied mainly to build the *Compressed Suffix Tree* [143] and *Wavelet Tree* [120].

Wavelet tree This is a CDS designed to answer `rank` and `select` queries over sequences of size n in an alphabet of size σ [120]. It is also used in computational geometry to represent points on a grid [41]. The application of parallel computing appears mainly in building the structure [99, 49, 48, 55, 94, 167, 50, 51, 61, 62, 149, 95, 59, 62, 150], but the literature explores parallel queries [99, 13, 156, 95]. Parallelism was applied in distributed memory models [49, 48, 13], external memory [55], shared memory [48, 94, 61, 62, 149, 95, 59, 62], Map Reduce [167], and bit-parallel instructions [99, 50, 51].

Compressed Suffix Tree A widely explored CDS in its parallel construction is the *Compressed Suffix Tree* [143]. This structure allows Pattern Matching queries on a compactly represented sequence. The structure consists of a Compressed Suffix Array [108], a Compressed LCP array, and a Compressed Suffix Tree topology [16, 15]. The literature presents many proposals for the parallel construction of a Compressed Suffix Array [95, 93, 78, 79, 164, 60, 82], an LCP-Array [78, 148, 46, 60], and the Compressed Suffix Tree topology [16, 15]. Parallelism has been explored on GPU [46, 164] and distributed memory [60].

k^2-tree There are few studies on parallel computing applications on the k^2-tree. Álvares-García *et al*. [8] explored the construction and queries of the structure in a parallel environment under the distributed computing model. Broß *et al*. [29] proposed a parallel construction of the k^2-tree. The proposal divides the set of m points into p subsets to

build a k^2-tree on each of the p processors and finally merges all the k^2-trees into the final structure.

Other structures Fuentes-Sepúlveda *et al*. [57] presented two algorithms for constructing and querying a *Succinct Tree* [75] on the shared memory model. This structure allows the representation of compact trees. The first algorithm requires $O(n)$ Work, $O(\log n)$ Depth, $O(n \log n)$ extra bits of space, and supports $O(\log n)$ queries time. The second algorithm improves the query time to $O(c)$ for a positive constant c. The algorithm requires $O\left(n + \frac{n}{\log^c n} \log\left(\frac{n}{\log^c n}\right) + c^c\right)$ Work, $O\left(c + \log\left(\frac{nc^c}{\log^c n}\right)\right)$ Depth, and $O(n \log n)$ extra bits of space.

Labeit *et al*. [95] proposed a parallel construction of the *FM Index* [56]. Their construction is based on their version of the parallel construction of the Suffix Array, which performs the computation of the *Burrows-Wheeler Transform* [64] necessary for constructing the structure.

Köppl *et al*. [90] presented a parallel algorithm for constructing a *Block Tree* [20]. It uses the factorization proposed by Shun and Zhao [151], which requires $O(n)$ work and $O(\log^2 n)$ depth.

6.3 Parallel construction of k^2-raster

The original k^2-raster construction follows a *top-down* approach [97, 98]. It begins with the entire raster and determines its minimum and maximum values, `rMin` and `rMax`, by comparing every cell. If both values are equal, the process terminates; otherwise, it performs a subdivision and iterates through each subraster, and each subraster is processed recursively. This method involves repeated comparisons within and across subrasters, increasing computational cost. In contrast, the *bottom-up* approach builds the structure starting from the individual cells, aggregating the minimum and maximum values of each subraster. This method performs only two comparisons per cell and $2k^2$ per subraster. However, it requires additional pruning steps to remove subrasters with the same values, which implies updating the minimum and maximum arrays. Multiple comparisons can be executed in parallel, such as computing minimum and maximum

values across subrasters at the same level or performing the pruning stage by recalculating and relocating values to their new positions.

Our proposal linearizes the two-dimensional raster into a one-dimensional array to enable subsequent data structure construction using a *bottom-up* approach, defining an efficient construction of a k^2-raster using a parallel approach on GPGPU. In this implementation, CUDA is employed on an Nvidia GPU.

The steps considered in our proposal are the following:

1. *Raster linearization in Z-order*. The input raster values, initially arranged in *row-major* order, are reordered into Z-order to improve spatial locality and enable parallel processing.

2. *Computation of minimum, maximum, and filter levels*. The algorithm uses every four consecutive values for each level to compute the `min` and `max` arrays. Subsequently, we compute the `filter` array such that $\texttt{filter}[i] = 0$ if $\texttt{min}[i] = \texttt{max}[i]$, and $\texttt{filter}[i] = 1$ otherwise. Furthermore, we compute a $\texttt{count}[i]$ array, representing the cumulative number of ones in $\texttt{filter}[1..i]$, analogous to a rank query. We organize the resulting arrays in a *bottom-up* order.

3. *Pruning the unnecessary nodes*. For this, we use `filter` and `count` arrays. We compute two additional arrays to facilitate the subsequent reorganization: the differences between nodes and their parents, and the offset for each level.

4. *Reordering of levels within the arrays*. Using the previously computed offsets, the arrays are transformed from *bottom-up* order to *top-down* order. This final reordering produces the arrays in the format consistent with a k^2-raster representation.

6.3.1 Raster linearization in Z-order

Raster data initially uses a *row-major* order, where the array is traversed row by row from top to bottom and from left to right. To transform this representation into Z-order, the algorithm computes a Morton number for each element by interleaving the bits of

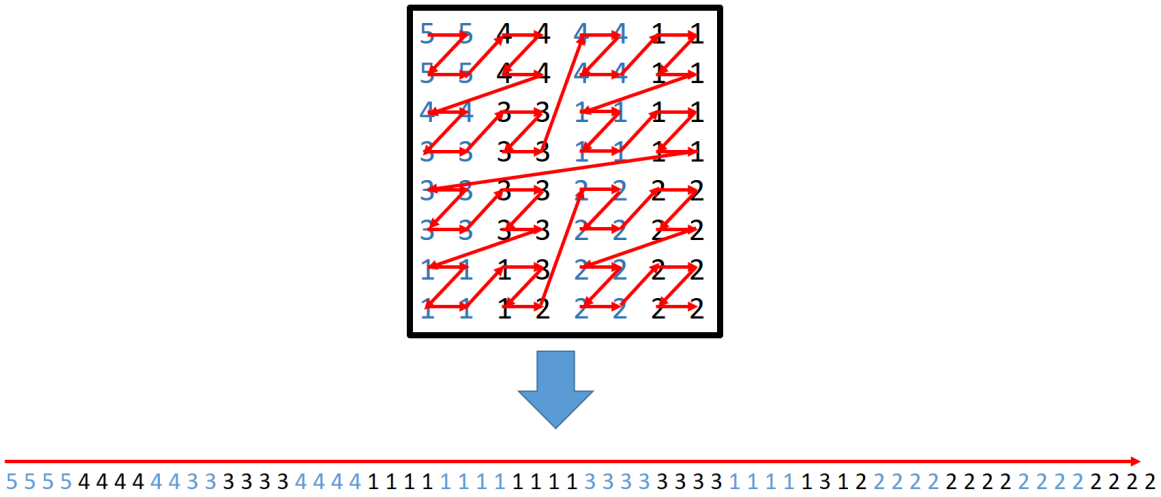


Figure 6.1: Example of a raster linearization to a Z-order

its row and column indices. The Morton number determines the position of the element in the Z-order array.

We consider two approaches for this linearization:

1. *Reading in Z-order*: For each value, the algorithm computes its Morton number from the row and column indices and directly stores it in the corresponding position of the Z-order array. On CPUs, we used the `PDEP` instruction[2], which accelerates this bit-interleaving operation.

2. *GPU ordering*: This approach first reads the raster in *row-major* order and then transforms it into Z-order on the GPU. The kernel launches n threads, where n corresponds to the number of elements in the array. Because GPUs do not support the `PDEP` instruction, we replace it with a function that performs bit interleaving[3] to compute Morton Numbers.

Figure 6.1 shows an example of raster linearization into Z-order.

[2] `https://www.felixcloutier.com/x86/pdep`
[3] `https://lemire.me/blog/2018/01/08/how-fast-can-you-bit-interleave-32-bit-integers/`

6.3.2 Computation of minimum, maximum, and filter levels

As the second step, the algorithm builds three arrays: the minimum array `min`, the maximum array `max`, and the binary array `filter`.

To construct the first level, a kernel launches $n/4$ threads. Each thread processes four contiguous elements in Z-order, computes the smallest and largest values, and stores them in the first level of `min` and `max`. This level, therefore, contains $n/4$ integers in each array. The `filter` stores 1 when the minimum and maximum differ, and 0 otherwise. This structure plays a central role in constructing the `Tree` array in the next stage.

For higher levels, the algorithm uses the previous level as input. The second level of `min` derives from the first level of `min`, and the second level of `max` derives from the first level of `max`. The second level of `filter` is generated similarly to the previous level. The kernel launches $n/16$ threads at this stage, each handling four entries, while the remaining threads remain idle.

The algorithm repeats this process until reaching a final level with four values. After completing each level, the algorithm executes a synchronization barrier to guarantee the complete construction of the level before advancing.

Figure 6.2 presents an example of this process. The example shows a Z-order array of 64 elements. Constructing the first level of `min`, `max`, and `filter` requires 16 threads. The second level requires four threads, while the remaining 12 remain unused. The second level contains only four values, so it becomes the last level and finishes the process.

Finally, to compute the `count` array, the algorithm applies the *exclusive scan* function available in the CUDA `thrust` library[4].

6.3.3 Pruning the unnecessary nodes

In this stage, the algorithm removes the nodes in `min`, `max`, and `filter` representing submatrices with repeated values. This process produces the intermediate arrays `inter_min`, `inter_max`, and `inter_filter`; intermediate structures that require the last

[4] `https://docs.nvidia.com/cuda/thrust/index.html`

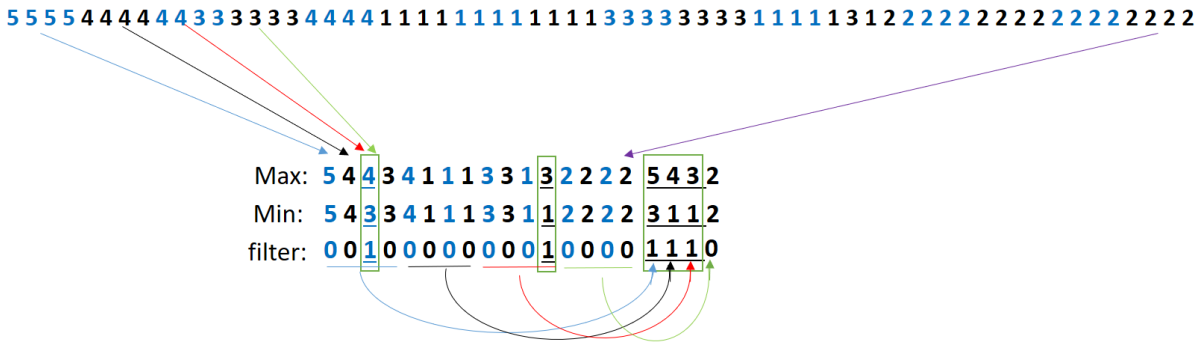


Figure 6.2: Example of construction of `min`, `max` and `filter` arrays

stage to evolve the final arrays that construct the k^2-raster.

For constructing the arrays of the k^2-raster, the algorithm traverses the arrays from bottom to top. At the first level, it constructs only `inter_max`, because this array preserves the submatrices that contain different individual cells. The algorithm accesses the first level of `max`, `filter`, and the Z-order array, selects the submatrices whose `filter` value is 1, and copies the difference between each submatrix and its representative into `inter_max`.

Figure 6.3 illustrates this step. In the example, two submatrices (3 and 12) contain four distinct elements. The algorithm computes their value in the first level of `max`, evaluates the difference concerning the four underlying elements in the Z-order array, and stores the result in `inter_max`. The process yields eight values corresponding to the lowest level of `inter_max`.

For the higher levels, the construction of `inter_max` follows similarity, replacing the Z-order array with the `max` array of the previous level. The construction of `inter_min` requires a comparison between the `min` arrays of the current and previous levels, and the `filter` array of the previous level. The algorithm considers only the positions where `filter` is 1, computes the difference between levels, and stores it in `inter_min`. Unlike `inter_max`, which groups values in fours, `inter_min` processes each element individually.

For `inter_filter`, the algorithm compares the `filter` arrays of the current and previous levels. It copies into `inter_filter` those values from the previous level whose representative in the current level equals 1.

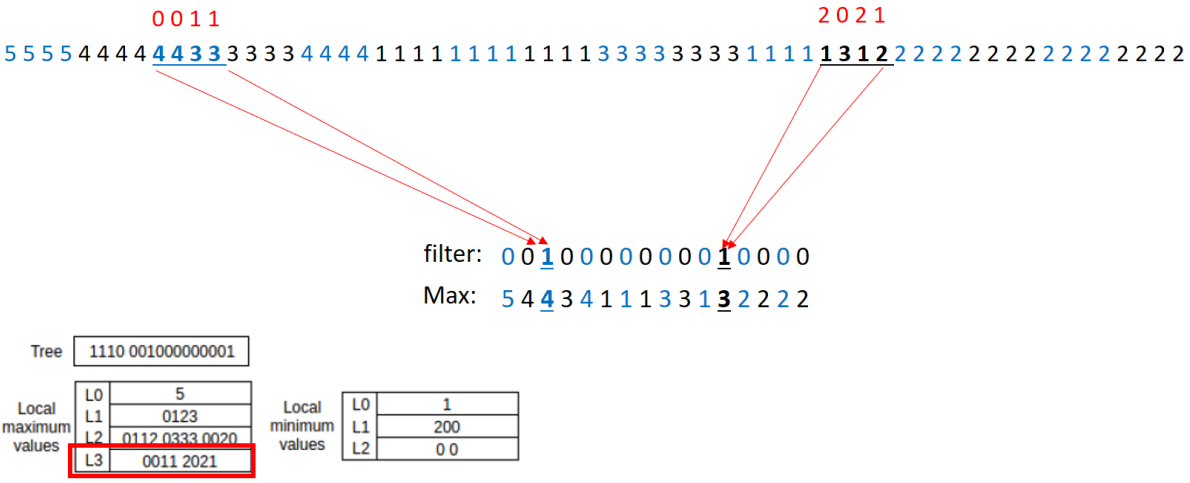


Figure 6.3: Construction of the third level of inter arrays

Figure 6.4 shows an example of this process. For `inter_min`, the algorithm uses the `filter` array of the previous level, selects the elements at positions 3 and 12, obtains their representatives in the upper level (positions 1 and 3), and stores their respective differences. For `inter_filter`, since the first three positions of the current `filter` array are 0, the algorithm copies the first three values (12 bits) of the previous `filter` array. At the last level, it copies the four bits of `filter` directly into `inter_filter`, as seen in Figure 6.5.

Finally, to reorder the intermediate arrays into the final arrays `Lmin`, `Lmax`, and `T`, the algorithm computes the number of elements in each level. For this purpose, it calculates three offset arrays for each intermediate structure, which indicate the starting position of every level. This computation runs in parallel. In the example, the offsets are $[0, 8, 19, 23]$ for `inter_max`, $[0, 0, 2, 5]$ for `inter_min`, and $[0, 0, 12, 16]$ for `inter_filter`.

6.3.4 Reordering of levels within the arrays

Using the offset arrays, the algorithm reorders the levels of the unpruned nodes in parallel to produce `Lmin`, `Lmax`, and `T`. The intermediate arrays `inter_min`, `inter_max`, and `inter_filter` store the levels in a *bottom-up* order, while the final structure requires a *top-down* order. To achieve this, the algorithm launches a kernel in which each thread receives a value from an intermediate array and places it in the corresponding position

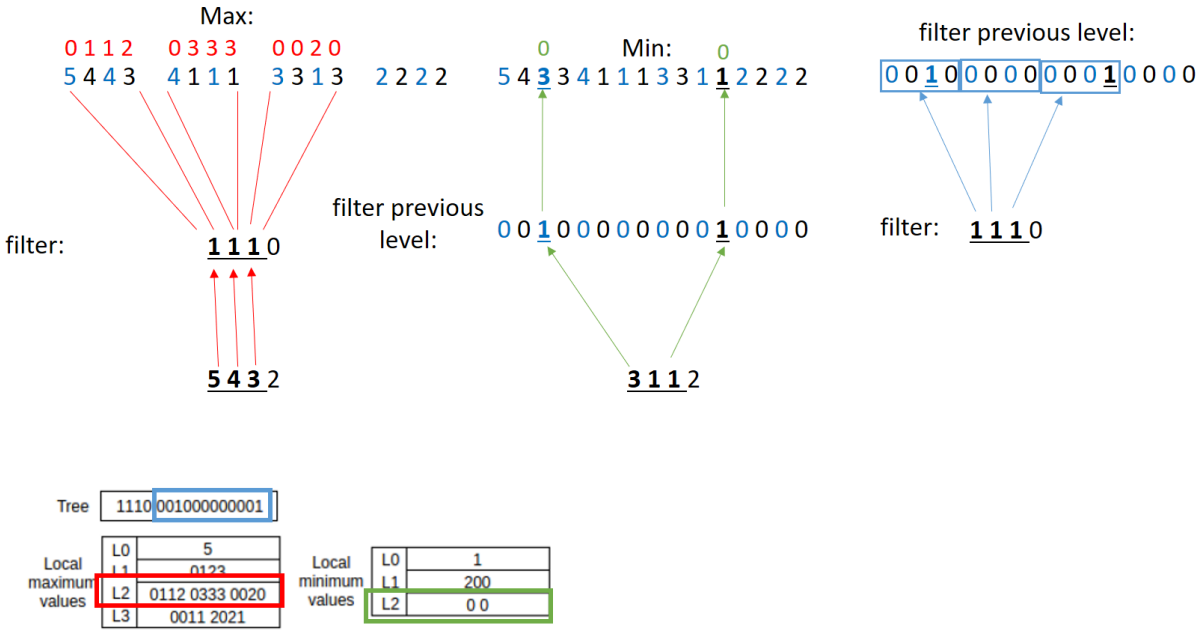


Figure 6.4: Construction of the second level of inter arrays

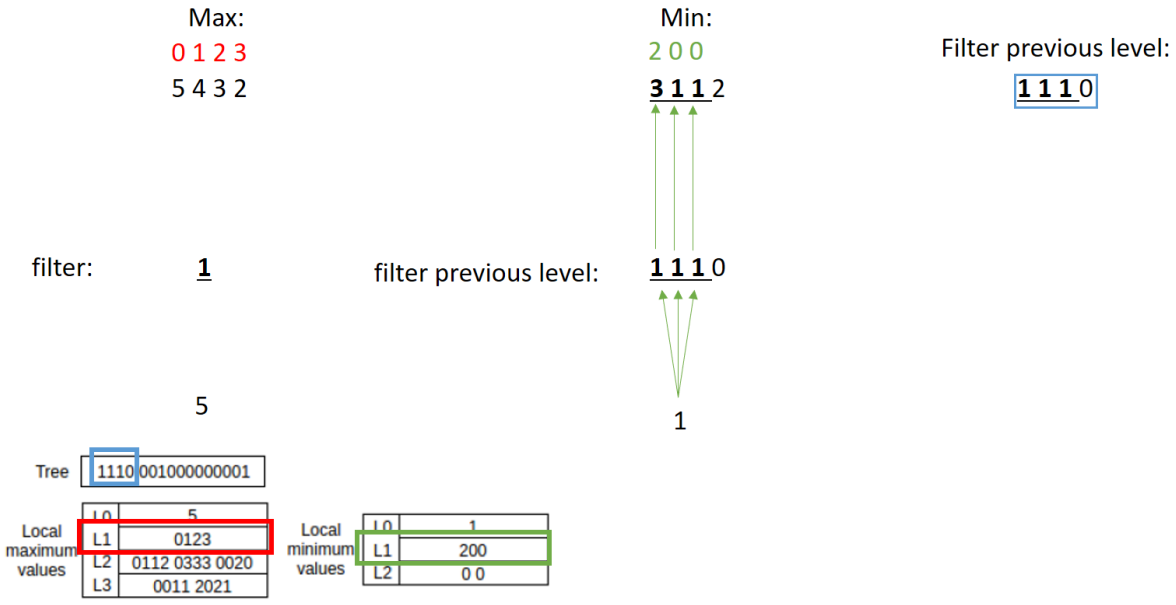


Figure 6.5: Construction of the first level of inter arrays

in the output array. Consequently, we also obtain the values `rMin` and `rMax`.

6.4 Experimental results

We conducted the experiments using the datasets described in [98], publicly available at `https://zenodo.org/records/17594578`. The datasets come from two sources. The first is the *Spanish Geographic Institute*, which provides terrain elevation data from Spain (Table 6.1). The second is *WorldClim*, which offers global climate data (Table 6.2).

Name	Amount datasets	#rows	#cols	#Different values
9-MDT05	1	3,881	5,841	227
7-MDT05	1	3,881	5,841	903
5-MDT05	1	3,881	5,841	3,606
3-MDT05	1	3,881	5,841	14,415
1-MDT05	1	3,881	5,841	57,586
MDT05	1	3,881	5,841	114,966

Table 6.1: Datasets from *Spanish Geographic Institute*

Name	Amount datasets	#rows	#cols	#Different values
eua-1X1	25	3,600	3,600	252
eua-2X2	16	7,200	7,200	413

Table 6.2: Datasets from *WorldClim*

We evaluated the following algorithms:

- `rec`: Corresponds to the original and recursive version presented in [97, 98][5].
- `seq`: Corresponds to the sequential version of our parallel proposal. This version follows a *bottom-up* approach, generating `Lmin`, `Lmax` and `T` such as 32 bits integer arrays and single bit array respectively. The original k^2-raster represents integer arrays in a DAC Vector [27] and the bit array in a bitmap [76]. Both structures

[5] `https://gitlab.lbd.org.es/fsilva/k2-raster`

are implemented in the SDSL library[6], which is not considered for parallelization in our study.

- `par`: Corresponds to the parallel version with CUDA, with the following variations: `par8` (8 threads per block), `par16` (16 threads per block), `par32` (32 threads per block), `par64` (64 threads per block), `par128` (128 threads per block), `par256` (256 threads per block), `par512` (512 threads per block) and `par1024` (1024 threads per block).

For each algorithm/variant and dataset, we executed the experiments 10 times and obtained the average execution time. Therefore, for the data in Table 6.1, the results correspond to the average of the 10 executions (each dataset contains one raster); for the data in Table 6.2, the results correspond to the average of 250 executions for `eua-1X1` (10 executions for each 25 rasters), and 160 executions for `eua-2X2` (10 executions for each 16 rasters). We executed experiments on a Nvidia GeForce GTX 1050 GPU with CUDA-10.1, which can compute 10,240 threads concurrently in total. For time measurement, we used the library Chrono from C++. The algorithms codes are available at `https://gitlab.com/mmunocan/k2raster-paralelo.git`.

Table 6.3 presents the execution times for constructing the k^2-raster from raster data in Row-Major. The `rec` algorithm exhibits the largest execution times due to the additional compression applied by the SDSL library. When excluding compression, the results highlight the impact of the number of threads in the `par` algorithm.

From `par64` onward, the parallel version consistently outperforms `seq` for most datasets. Its speedup reaches its maximum values for 64 threads per block, with a 1.47 for `eua-2X2` (see Table 6.4). Although these results are considered modest within the context of massively parallel computing, they provide an interesting proof of concept that exhibits the potential of this approach.

Tables 6.5 and 6.6 show the execution times for raster reading, linearization in Z-order, and structure construction using different techniques. Results remain stable at approximately 500 ms regardless of whether we apply `pdep` or bit interleaving, and independently of the number of threads.

[6] `https://github.com/simongog/sdsl-lite`

Dataset	Sequential		Parallel							
	rec	seq	par8	par16	par32	par64	par128	par256	par512	par1024
9-MDT05	1828.5	223.8	239.4	187.2	164.3	154.4	154.0	153.2	153.0	153.1
7-MDT05	2646.7	284.8	352.9	259.0	215.5	196.3	195.9	194.9	194.4	194.0
5-MDT05	3495.3	301.7	433.4	309.2	250.3	224.5	223.6	222.4	221.7	221.4
3-MDT05	3709.7	301.9	451.0	319.8	257.7	230.9	229.6	228.2	227.7	227.2
1-MDT05	3998.8	303.3	452.2	320.5	258.3	231.4	229.8	229.0	228.1	227.5
MDT05	4200.2	303.3	453.2	320.9	258.4	231.3	230.5	229.4	228.3	227.8
eua-1X1	1164.8	68.7	125.0	102.9	92.9	88.6	88.4	88.3	88.1	88.1
eua-2X2	4422.4	295.1	367.3	266.6	220.5	200.5	199.8	198.8	198.3	198.0

Table 6.3: Execution time in $[ms]$

Dataset	par8	par16	par32	par64	par128	par256	par512	par1024
9-MDT05	0.93	1.20	1.36	1.45	1.45	1.46	1.46	1.46
7-MDT05	0.81	1.10	1.32	1.45	1.45	1.46	1.47	1.47
5-MDT05	0.70	0.98	1.21	1.34	1.35	1.36	1.36	1.36
3-MDT05	0.67	0.94	1.17	1.31	1.32	1.33	1.33	1.33
1-MDT05	0.67	0.95	1.17	1.31	1.32	1.32	1.33	1.33
MDT05	0.67	0.95	1.17	1.31	1.32	1.32	1.33	1.33
eua-1X1	0.55	0.67	0.74	0.78	0.78	0.78	0.78	0.78
eua-2X2	0.80	1.11	1.34	1.47	1.48	1.48	1.49	1.49

Table 6.4: Speedup for construction time

		Parallel							
Dataset	seq	par8	par16	par32	par64	par128	par256	par512	par1024
9-MDT05	551.9	549.9	550.5	549.4	550.2	549.7	549.7	549.6	549.2
7-MDT05	549.4	549.2	548.7	549.1	549.5	549.6	549.8	549.4	549.4
5-MDT05	549.5	549.3	550.2	550.8	549.3	549.7	550.2	550.6	550.4
3-MDT05	550.6	550.5	550.5	550.4	550.5	549.9	550.4	551.0	549.7
1-MDT05	551.8	551.0	552.1	551.1	551.8	552.3	551.6	607.9	551.0
MDT05	549.6	548.6	550.1	550.0	549.9	550.0	550.0	549.6	550.3
eua-1X1	237.5	238.5	238.9	239.0	238.9	239.0	239.8	239.0	239.0
eua-2X2	948.1	950.5	950.6	951.2	951.1	951.5	951.2	951.1	957.5

Table 6.5: Execution time for Z-order reading with pdep in $[ms]$

The results show that massively parallel computation leads to a moderate reduction in the k^2-raster construction time (excluding data loading). Although the achieved speedup remains limited, the results highlight the potential of this approach and point to clear opportunities for further optimization. Concerning linearization, both techniques yield comparable performance, with no significant differences.

6.5 Conclusions and future work

This chapter presents a parallel algorithm for constructing the Compact Data Structure k^2-raster using massively parallel computing on a GPGPU. The algorithm contains four phases: raster linearization in Z-order, computation of minimum, maximum, and filter levels, pruning of unnecessary nodes, and reordering of levels within the arrays. The

Dataset	seq	Parallel par8	par16	par32	par64	par128	par256	par512	par1024
9-MDT05	550.7	550.7	550.6	550.2	550.6	550.4	550.8	550.4	550.6
7-MDT05	549.8	549.6	550.2	550.0	550.9	549.9	550.2	550.7	550.3
5-MDT05	550.4	550.1	550.3	550.1	550.7	550.3	550.5	550.5	550.4
3-MDT05	550.2	549.2	551.0	550.0	550.7	550.9	550.6	550.3	550.4
1-MDT05	551.5	552.3	551.9	551.9	551.9	552.2	552.5	552.4	552.4
MDT05	549.1	550.2	549.7	549.5	548.9	550.0	551.5	550.1	552.1
eua-1X1	236.0	238.5	239.5	238.9	238.8	238.9	238.9	238.9	238.9
eua-2X2	940.5	951.0	951.1	951.2	951.4	951.7	951.3	950.9	951.1

Table 6.6: Execution time for Z-order reading with bit interleaving linearization in $[ms]$

algorithm was implemented using CUDA on an Nvidia GPU.

The evaluation compares four algorithms: the original recursive version (rec); the sequential version of our proposal (seq); and the parallel variants (par8–par1024), which differ in the number of threads per block. The results show that rec shows the highest execution time due to the additional compression stage performed by the SDSL library. In contrast, seq and par exclude this library, achieving faster execution. When comparing seq with the parallel versions, par32 achieves performance comparable to the sequential implementation, and increasing the number of threads per block further reduces construction time. Its speedup reaches its maximum values for 64 threads per block, with a 1.47, a middling value considering the context of massively parallel computing. The main bottleneck likely stems from the continuous memory allocation during Z-order linearization and the waiting time required to complete one level before initiating the next one. The k^2-raster is not inherently optimized for GPU memory architectures. Nevertheless, the results highlight the advantages of applying massively parallel computing to the k^2-raster construction process, providing a proof of concept that demonstrates the effectiveness of this approach.

Furthermore, we evaluated the raster linearization step using two techniques: *bit interleaving* and pdep. The results show no significant differences, indicating that both methods achieve similar efficiency in obtaining the linearized raster from the input data.

As part of our future work, we plan to conduct a study analyzing the SDSL library to identify opportunities for parallelization within its implemented structures. In particular,

we will focus on constructing the bitmap and the DAC vector, which are the underlying Compact Data Structures required to construct the k^2-raster fully.

Chapter 7

Conclusions

This chapter presents the main conclusions of the thesis and outlines potential directions for future research. Section 7.1 summarizes the conclusions, while Section 7.2 discusses future work.

7.1 Conclusions

This thesis uses different approaches to explore raster and raster time series compression, using mainly Compact Data Structures (CDS), specifically the k^2-raster for representing rasters and the Heuristic T-k^2-raster for representing raster time series. The contributions include an evaluation of the effectiveness of the heuristic applied in the T-k^2-raster based on dynamic programming (DP), a new heuristic proposal for the T-k^2-raster based on clustering, an extensive study of raster compression based on compressibility measures, a new CDS for raster times series representation based on linearization and the k^2-raster, and a new algorithm for the k^2-raster construction using massively parallel computing.

Based on the work developed in this thesis, the results confirm the main hypothesis, but considering certain circumstances discovered throughout the research. Improving the compression and efficiency of CDS algorithms applied to raster data is possible. The circumstances found throughout this thesis are guided by its specific goals, from which we derived the following results and conclusions:

To improve compression of raster time series indexed on the T-k^2-raster applying dynamic programming and clustering methods.

- Behind evaluating the Heuristic T-k^2-raster, we observed no significant difference in compression performance compared to the DP approach on real data,

demonstrating that the heuristic achieves a near-optimal compression level. However, there is a considerable difference between the heuristic and DP on synthetic data when there is high temporal locality and low spatial locality. In the last scenario, enhancing the heuristic to increase the compression of a raster time series is possible.

- The clustering proposal, which presents an alternative heuristic to the T-k^2-raster, overcoming certain limitations of the original heuristic, shows similar results to those of the Heuristic T-k^2-raster on real data, demonstrating that it is as competitive as the original heuristic. However, clustering presents a significant advantage in synthetic data with rasters repeated in cycles.

- The proposed structure, called ZT-k^2-raster, transforms the temporal locality of a raster time series into spatial locality, to index it into a k^2-raster. The results show that our proposal is competitive regarding compression capability, but not in terms of construction and query time.

To evaluate raster compression via compressibility measures and propose a new compressibility measure applied to rasters.

- The evaluation of raster compressibility measures revealed a high positive correlation between the compressibility measures and raster size after applying different compressors and CDS, as well as a high negative correlation between the compressibility measures results and spatial locality. The influence of noise on the studied measures was also described, with the number of affected cells being the most significant factor.

- The proposed compressibility measure, δ_Δ, is influenced by two critical factors: the raster alphabet and the frequency distribution of its symbols. The measure increases as the alphabet size and the skewness of the frequency distribution increase. δ_Δ is a good predictor of compressors with similar behavior, such as NetCDF and the k^2-raster.

To design and implement parallel algorithms for constructing a CDS for raster data.

- After designing and implementing a parallel algorithm to construct the k^2-raster, the results indicated the potential benefits of applying massively parallel algorithms on GPGPUs. Compared to the non-parallel version, the construction time decreases as the number of threads per block increases, although the speedup remains moderate. However, the evaluation process excluded the parallel construction of advanced compression-enhancing structures offered by the SDSL library.

To analyze raster compression and query performance improvements on scientific images.

- This thesis used scientific images within a real-world dataset collection that contains five raster time series representing a mouse liver tissue cells. Each raster time series corresponds to a different protein from the same tissue [113]. Those images consider a range of the lowest values to represent noise. Behind applying a low filtering percentage (less than 10%), which discards the smallest values, the results showed a significant improvement in raster time series compression using the T-k^2-raster and the ZT-k^2-raster.

7.2 Future Work

This thesis leaves an open line for future work in multiple paths:

- Developing a machine learning model to improve raster data compression is promising. Learned indices [3] enable compression by finding and exploiting intrinsic data patterns. To achieve compression, they can find additional patterns beyond the data locality and ciclicity of rasters and raster time series.

- The $\delta_{\triangle}$ measure is an interesting proposal to explore further. A theoretical analysis of the measures may determine interesting properties.

- Regarding the analysis of compressibility measures, although we present a study of their application to raster time series, it is possible to expand it in greater depth. In particular, studying other linearization functions explicitly designed for three-dimensional data is reasonable, such as the Hilbert 3D curve [77].

- The results of parallelizing the construction of the k^2-raster pave the way for further parallelization of the entire structure, including the SDSL library. Subsequently, it is possible to parallelize the structure construction process fully, thus expanding the advantages of parallel computing.

Bibliography

[1] Arezoo Abdollahi, Neil Bruce, Shahin Kamali, and Rezaul Karim. Lossless image compression using list update algorithms. In *SPIRE*, pages 16–34, Cham, 2019. Springer International Publishing.

[2] Amin Aghaee, Pejman Shamsipour, Shawn Hood, and Rasmus Haugaard. A convolutional neural network for semi-automated lineament detection and vectorisation of remote sensing data using probabilistic clustering: A method and a challenge. *Computers & Geosciences*, 151:104724, 2021.

[3] Abdullah Al-Mamun, Hao Wu, and Walid G. Aref. A tutorial on learned multi-dimensional indexes. In *SIGSPATIAL*, SIGSPATIAL '20, page 1–4, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Osama K Al-Shaykh and Russell M Mersereau. Lossy compression of noisy images. *IEEE Transactions on Image Processing*, 7(12):1641–1652, 1998.

[5] Khawla Ali Abd Al-Hameed. Spearman's correlation coefficient in statistical analysis. *International Journal of Nonlinear Analysis and Applications*, 13(1):3249–3255, 2022.

[6] Mazin Alkathiri, Jhummarwala Abdul, and M. B. Potdar. Kluster: Application of k-means clustering to multidimensional geo-spatial data. In *ICICIC*, pages 1–7, Piscataway, New Jersey, United States, 8 2017. IEEExplore.

[7] Mazin Alkathiri, Abdul Jhummarwala, and MB Potdar. Multi-dimensional geospatial data mining in a distributed environment using mapreduce. *Journal of Big Data*, 6(1):82, 2019.

[8] Sandra Álvarez-García, Nieves R. Brisaboa, Carlos Gómez-Pantoja, and Mauricio Marin. Distributed query processing on compressed graphs using k2-trees. In *SPIRE*, pages 298–310, Cham, 2013. Springer International Publishing.

[9] Sandra Alvarez-Garcia, Guillermo de Bernardo, Nieves R. Brisaboa, and Gonzalo Navarro. A succinct data structure for self-indexing ternary relations. *Journal of Discrete Algorithms*, 43:38–53, 2017.

[10] Esra Alzaghoul, Mohammad Belal Al-Zoubi, Ruba Obiedat, and Fawaz Alzaghoul. Applying machine learning to dem raster images. *Technologies*, 9(4):87, 2021.

[11] Rajaraman Anand and Ullman Jeffrey David. *Mining of massive datasets*. Cambridge University Press, Cambridge, England, 2011.

[12] Alberto Apostolico. The myriad virtues of subword trees. In *Combinatorial Algorithms on Words*, pages 85–96, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[13] Diego Arroyuelo, Veronica Gil-Costa, Senén González, Mauricio Marin, and Mauricio Oyarzún. Distributed search based on self-indexed compressed text. *Information Processing & Management*, 48(5):819–827, 2012. Large-Scale and Distributed Systems for Information Retrieval.

[14] David Arthur and Sergei Vassilvitskii. K-means++ the advantages of careful seeding. In *SODA*, pages 1027–1035, New York, NY, USA, 1 2007. Association for Computing Machinery.

[15] Uwe Baier, Timo Beller, and Enno Ohlebusch. Parallel construction of succinct representations of suffix tree topologies. In *SPIRE*, pages 234–245, Cham, 2015. Springer International Publishing.

[16] Uwe Baier, Timo Beller, and Enno Ohlebusch. Space-efficient parallel construction of succinct representations of suffix tree topologies. *ACM J. Exp. Algorithmics*, 22, January 2017.

[17] Hideo Bannai, Dominik Köppl, and Zsuzsanna Lipták. A Survey of the Bijective Burrows-Wheeler Transform. In *OASIcs*, volume 131 of *Open Access Series in Informatics (OASIcs)*, pages 2:1–2:26, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[18] Djamal Belazzougui and Fabio Cunial. Fast label extraction in the cdawg. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *SPIRE*, pages 161–175, Cham, 2017. Springer International Publishing.

[19] Djamal Belazzougui, Fabio Cunial, Travis Gagie, Nicola Prezza, and Mathieu Raffinot. Composite repetition-aware data structures. In *CPM*, pages 26–39, Cham, 2015. Springer International Publishing.

[20] Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez, Simon J. Puglisi, and Yasuo Tabei. Block trees. *Journal of Computer and System Sciences*, 117:1–22, 2021.

[21] A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, jul 1987.

[22] Nieves R. Brisaboa, Guillermo de Bernardo, Gilberto Gutiérrez, Miguel R. Luaces, and José R. Paramá. Efficiently Querying Vector and Raster Data. *The Computer Journal*, 60(9):1395–1413, 02 2017.

[23] Nieves R. Brisaboa, Ana Cerdeira-Pena, Guillermo de Bernardo, Gonzalo Navarro, and Óscar Pedreira. Extending general compact querieable representations to gis applications. *Information Sciences*, 506:196–216, 2020.

[24] Nieves R. Brisaboa, Guillermo de Bernardo, Roberto Konow, and Gonzalo Navarro. K2-treaps: Range top-k queries in compact space. In *SPIRE*, pages 215–226, Cham, 2014. Springer International Publishing.

[25] Nieves R Brisaboa, Travis Gagie, Adrián Gómez-Brandón, and Gonzalo Navarro. Two-dimensional block trees. *The Computer Journal*, 67(1):391–406, 12 2022.

[26] Nieves R. Brisaboa, Susana Ladra, and Gonzalo Navarro. k2-trees for compact web graph representation. In *SPIRE*, pages 18–30, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[27] Nieves R Brisaboa, Susana Ladra, and Gonzalo Navarro. DACs: Bringing direct access to variable-length codes. *Information Processing & Management*, 49(1):392–404, 2013.

[28] Nieves R. Brisaboa, Susana Ladra, and Gonzalo Navarro. Compact representation of web graphs with extended functionality. *Information Systems*, 39:152–174, 2014.

[29] Jan Broß, Simon Gog, Matthias Hauck, and Marcus Paradies. Fast construction of compressed web graphs. In *SPIRE*, pages 116–128, Cham, 2017. Springer International Publishing.

[30] Mónica Caniupán, Rodrigo Torres-Avilés, Tatiana Gutiérrez-Bunster, and Manuel Lepe. Efficient computation of map algebra over raster data stored in the k2-acc compact data structure. *GeoInformatica*, 26(1):95–123, 2022.

[31] Lorenzo Carfagna and Giovanni Manzini. The landscape of compressibility measures for two-dimensional data. *IEEE Access*, 12:87268–87283, 2024.

[32] Lorenzo Carfagna, Giovanni Manzini, Giuseppe Romana, Marinella Sciortino, and Cristian Urbina. Generalization of repetitiveness measures for two-dimensional strings. In *SPIRE*, pages 57–72, Cham, 2025. Springer Nature Switzerland.

[33] Ana Cerdeira-Pena, Guillermo de Bernardo, Antonio Fariña, José Ramón Paramá, and Fernando Silva-Coira. Towards a compact representation of temporal rasters. In *SPIRE*, pages 117–130, Cham, 2018. Springer International Publishing.

[34] M. Charikar, E. Lehman, Ding Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

[35] Kevin Chow. *Compact data structures for remote sensing data*. PhD thesis, Universitat de Barcelona, 2022.

[36] Kevin Chow, Dion Eustathios Olivier Tzamarias, Ian Blanes, and Joan Serra-Sagristà. Using predictive and differential methods with k2-raster compact data structure for hyperspectral image lossless compression. *Remote Sensing*, 11(21), 2019.

[37] Kevin Chow, Dion Eustathios Olivier Tzamarias, Miguel Hernández-Cabronero, Ian Blanes, and Joan Serra-Sagristà. Analysis of variable-length codes for integer encoding in hyperspectral data compression with the k2-raster compact data structure. *Remote Sensing*, 12(12), 2020.

[38] Kevin Chow, Dion Eustathios Olivier Tzarmarias, Miguel Hernández-Cabronero, Ian Blanes, and Joan Serra-Sagristà. Performance improvement on k²-raster compact data structure for hyperspectral scenes. *IEEE Geoscience and Remote Sensing Letters*, 19:1–5, 2022.

[39] Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Algorithms*, 17(1), dec 2021.

[40] David Clark. *Compact pat trees*. PhD thesis, University of Waterloo, 1997.

[41] Francisco Claude and Gonzalo Navarro. The wavelet matrix. In *SPIRE*, pages 167–179, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[42] Francisco Claude, Patrick K. Nicholson, and Diego Seco. On the compression of search trees. *Information Processing & Management*, 50(2):272–283, 2014.

[43] Nataly Cruces, Diego Seco, and Gilberto Guitérrez. A compact representation of raster time series. In *DCC*, pages 103–111, 2019.

[44] Guillermo de Bernardo, Sandra Álvarez-García, Nieves R. Brisaboa, Gonzalo Navarro, and Oscar Pedreira. Compact querieable representations of raster data. In *SPIRE*, pages 96–108, Cham, 2013. Springer International Publishing.

[45] Oscar Plaza de los Reyes, Mónica Caniupán, Rodrigo Torres-Avilés, and Tatiana Gutiérrez-Bunster. Map algebra algorithms over raster data stored in the k2-raster compact data structure. In *SCCC*, pages 1–4, 2022.

[46] Mrinal Deo and Sean Keely. Parallel suffix array and least common prefix for the gpu. *SIGPLAN Not.*, 48(8):197–206, February 2013.

[47] P. DEUTSCH. Deflate compressed data format specification version 1.3. *IETF RFC 1951*, 1996.

[48] Patrick Dinklage, Jonas Ellert, Johannes Fischer, Florian Kurpicz, and Marvin Löbel. Practical wavelet tree construction. *ACM J. Exp. Algorithmics*, 26, July 2021.

[49] Patrick Dinklage, Johannes Fischer, and Florian Kurpicz. *Constructing the Wavelet Tree and Wavelet Matrix in Distributed Memory*, pages 214–228. SIAM, 2020.

[50] Patrick Dinklage, Johannes Fischer, Florian Kurpicz, and Jan-Philipp Tarnowski. Bit-parallel (compressed) wavelet tree construction. In *DCC*, pages 81–90, 2023.

[51] Patrick Dinklage, Johannes Fischer, Florian Kurpicz, and Jan-Philipp Tarnowski. Bit-parallel wavelet tree construction (abstract). In *Proceedings of the 2024 ACM Workshop on Highlights of Parallel Computing*, HOPC'24, page 37–38, New York, NY, USA, 2024. HOPC.

[52] Luana Pereira dos Reis and Daniel S Kaster. A compact and efficient data structure for line-based processing of series. In *ADBIS*, page 25. Springer Nature, 2024.

[53] Luana Pereira dos Reis and Daniel S. Kaster. A compact and efficient data structure for line-based processing of series of raster data. In *ADBIS*, pages 25–34, Cham, 2025. Springer Nature Switzerland.

[54] Matt Duckham, Qian Chayn Sun, and Michael F Worboys. *GIS: a computing perspective*. CRC press, 2023.

[55] Jonas Ellert and Florian Kurpicz. Parallel external memory wavelet tree and wavelet matrix construction. In *SPIRE*, pages 392–406, Cham, 2019. Springer International Publishing.

[56] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *FOCS*, pages 390–398, 2000.

[57] Leo Ferres, José Fuentes-Sepúlveda, Meng He, and Norbert Zeh. Parallel construction of succinct trees. In *SEA*, pages 3–14, Cham, 2015. Springer International Publishing.

[58] Johannes Fischer, Tomohiro I, Dominik Köppl, and Kunihiko Sadakane. Lempel–ziv factorization powered by space efficient suffix trees. *Algorithmica*, 80:2048–2081, 2018.

[59] Johannes Fischer, Florian Kurpicz, and Marvin Löbel. *Simple, Fast and Lightweight Parallel Wavelet Tree Construction*, pages 9–20. SIAM, 2018.

[60] Patrick Flick and Srinivas Aluru. Parallel distributed memory construction of suffix and longest common prefix arrays. In *SC*, SC '15, New York, NY, USA, 2015. Association for Computing Machinery.

[61] José Fuentes-Sepúlveda, Erick Elejalde, Leo Ferres, and Diego Seco. Efficient wavelet tree construction and querying for multicore architectures. In *SEA*, pages 150–161, Cham, 2014. Springer International Publishing.

[62] José Fuentes-Sepúlveda, Erick Elejalde, Leo Ferres, and Diego Seco. Parallel construction of wavelet trees on multicore architectures. *Knowledge and Information Systems*, 51(3):1043–1066, 2017.

[63] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. On the approximation ratio of lempel-ziv parsing. In *LATIN*, pages 490–503, Cham, 2018. Springer International Publishing.

[64] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in bwt-runs bounded space. *J. ACM*, 67(1), jan 2020.

[65] Arthur Getis. *Spatial Autocorrelation*, pages 255–278. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[66] Mohsen Ghaffari and Krzysztof Nowicki. Massively parallel algorithms for minimum cut. In *PODC*, PODC '20, page 119–128, New York, NY, USA, 2020. Association for Computing Machinery.

[67] Sara Giuliani, Shunsuke Inenaga, Zsuzsanna Lipták, Giuseppe Romana, Marinella Sciortino, and Cristian Urbina. Bit catastrophes for the burrows-wheeler transform. *Theory of Computing Systems*, 69(2):19, 2025.

[68] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.

[69] Solomon Golomb. Run-length encodings (corresp.). *IEEE transactions on information theory*, 12(3):399–401, 1966.

[70] Keisuke Goto and Hideo Bannai. Simpler and faster lempel ziv factorization. In *DCC*, pages 133–142, 2013.

[71] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[72] Robert J Hijmans, Susan E Cameron, Juan L Parra, Peter G Jones, and Andy Jarvis. Very high resolution interpolated climate surfaces for global land areas. *International Journal of Climatology: A Journal of the Royal Meteorological Society*, 25(15):1965–1978, 2005.

[73] D Hilbcrt. Uebcr stetige abbildung einer linie auf ein flachenstuck'. *Math. Annal*, 38:459, 1891.

[74] A.J. Hussain, Ali Al-Fayadh, and Naeem Radi. Image compression techniques: A survey in lossless and lossy algorithms. *Neurocomputing*, 300:44–69, 2018.

[75] G. Jacobson. Space-efficient static trees and graphs. In *FOCS*, pages 549–554, Los Alamitos, CA, USA, nov 1989. IEEE Computer Society.

[76] Guy Joseph Jacobson. *Succinct static data structures*. Carnegie Mellon University, 1988.

[77] Lianyin Jia, Binbin Liang, Mengjuan Li, Yong Liu, Yinong Chen, and Jiaman Ding. Efficient 3d hilbert curve encoding and decoding algorithms. *Chinese Journal of Electronics*, 31(2):277–284, 2022.

[78] Juha Kärkkäinen and Peter Sanders. Simple linear work suffix array construction. In *ICALP*, pages 943–955, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[79] Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, November 2006.

[80] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *CPM*, pages 181–192, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[81] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *STOC*, STOC 2018, page 827–840, New York, NY, USA, 2018. Association for Computing Machinery.

[82] Jamshed Khan, Tobias Rubel, Erin Molloy, Laxman Dhulipala, and Rob Patro. Fast, parallel, and cache-friendly suffix array construction. *Algorithms for Molecular Biology*, 19(1):16, 2024.

[83] Kamran Khan, Saif Ur Rehman, Kamran Aziz, Simon Fong, and Sababady Sarasvady. Dbscan: Past, present and future. In *ICADIWT*, pages 232–238, Piscataway, New Jersey, United States, 2 2014. IEEExplore.

[84] Takuya Kida, Tetsuya Matsumoto, Yusuke Shibata, Masayuki Takeda, Ayumi Shinohara, and Setsuo Arikawa. Collage system: a unifying framework for compressed pattern matching. *Theoretical Computer Science*, 298(1):253–272, 2003. Selected Papers in honour of Setsuo Arikawa.

[85] J.C. Kieffer and En-Hui Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000.

[86] Dong Kyue Kim, Joong Chae Na, Jeong Seop Sim, and Kunsoo Park. Linear-time construction of two-dimensional suffix trees. *Algorithmica*, 59:269–297, 2011.

[87] R. Uday Kiran. Discovering knowledge hidden in raster images using rasterminer. In *ICDAR*, ICDAR '21, page 1, New York, NY, USA, 11 2021. Association for Computing Machinery.

[88] Tomasz Kociumaka, Gonzalo Navarro, and Francisco Olivares. Near-optimal search time in δ-optimal space, and vice versa. *Algorithmica*, 86(4):1031–1056, 2024.

[89] Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Toward a definitive compressibility measure for repetitive sequences. *IEEE Transactions on Information Theory*, 69(4):2074–2092, 2023.

[90] Dominik Köppl, Florian Kurpicz, and Daniel Meyer. Faster block tree construction. In *ESA*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[91] Oliver Kramer. *Machine Learning for Evolution Strategies*. Springer,Cham, 2016.

[92] Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theoretical Computer Science*, 483:115–133, 2013. Special Issue Combinatorial Pattern Matching 2011.

[93] Fabian Kulla and Peter Sanders. Scalable parallel suffix array construction. *Parallel Computing*, 33(9):605–612, 2007. Selected Papers from EuroPVM/MPI 2006.

[94] Dharmendra Kumar, Divakar Yadav, and Diwakar Singh Yadav. Spatial indexing and searching using parallel wavelet tree. *International Journal of Information Technology*, 15(7):3583–3591, 2023.

[95] Julian Labeit, Julian Shun, and Guy E. Blelloch. Parallel lightweight wavelet tree, suffix array and FM-index construction. *Journal of Discrete Algorithms*, 43:2–17, 2017.

[96] Susana Ladra. *Algorithms and compressed data structures for information retrieval*. PhD thesis, Universidade Da Coruña, 2011.

[97] Susana Ladra, José R. Paramá, and Fernando Silva-Coira. Compact and queryable representation of raster datasets. In *SSDBM*, SSDBM '16, New York, NY, USA, 2016. Association for Computing Machinery.

[98] Susana Ladra, José R. Paramá, and Fernando Silva-Coira. Scalable and queryable compressed storage structure for raster data. *Information Systems*, 72:179–204, 2017.

[99] Susana Ladra, Oscar Pedreira, Jose Duato, and Nieves R. Brisaboa. Exploiting simd instructions in current processors to improve classical string algorithms. In *ADBIS*, pages 254–267, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[100] N.J. Larsson and A. Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.

[101] Choonghwan Lee, MuQun Yang, and Ruth Aydt. Netcdf-4 performance report. In *Technical Report*. Technical report, HDF Group, 2008.

[102] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.

[103] Yansheng Li, Jiayi Ma, and Yongjun Zhang. Image retrieval from remote sensing big data: A survey. *Information Fusion*, 67:94–115, 2021.

[104] Feng Liu, Miguel Hernandez-Cabronero, Victor Sanchez, Michael W Marcellin, and Ali Bilgin. The current role of image compression standards in medical imaging. *Information*, 8(4):131, 2017.

[105] Òscar Maireles-González, Joan Bartrina-Rapesta, Miguel Hernández-Cabronero, and Joan Serra-Sagristà. Efficient lossless compression of integer astronomical data. *Publications of the Astronomical Society of the Pacific*, 135(1051):094502, 2023.

[106] Òscar Maireles-González, Joan Bartrina-Rapesta, Miguel Hernández-Cabronero, and Joan Serra-Sagristà. Analysis of lossless compressors applied to integer and floating-point astronomical data. In *DCC*, pages 389–398, 2022.

[107] U. MANBER. A new method for on-line string searches. *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithm, 1990*, 1990.

[108] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[109] Antonio Mariani, Italo Epicoco, Massimo Cafaro, and Marco Pulimeno. Grid-based contraction clustering in a peer-to-peer network. In *LOD*, pages 373–387, Berlin, Germany, 9 2023. Springer.

[110] José M Merigó. Using the probabilistic weighted average in decision making with distance measures. In *WCE*, volume 1, pages 1–4, Kwun Tong, Kowloon, Hong Kong, 6 2010. International Association of Engineers.

[111] Jose M Merigo and Montserrat Casanovas. Decision making with distance measures and linguistic aggregation operators. *International Journal of Fuzzy Systems*, 12(3):190–198, 2010.

[112] Alistair Moffat. Huffman coding. *ACM Computing Surveys (CSUR)*, 52(4), aug 2019.

[113] Hernán Morales-Navarrete, Fabián Segovia-Miranda, Piotr Klukowski, Kirstin Meyer, Hidenori Nonaka, Giovanni Marsico, Mikhail Chernykh, Alexander Kalaidzidis, Marino Zerial, and Yannis Kalaidzidis. A versatile pipeline for the multi-scale digital reconstruction and quantitative analysis of 3d tissue architecture. *eLife*, 4:e11214, dec 2015.

[114] Patrick AP Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37(1/2):17–23, 1950.

[115] G.M. Morton. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. International Business Machines Company, 1966.

[116] J. Ian Munro. Tables. In *FSTTCS*, pages 37–42, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[117] Martita Muñoz. Rasters Datasets, 7 2025.

[118] Martita Muñoz, José Fuentes-Sepúlveda, Cecilia Hernández, Gonzalo Navarro, Diego Seco, and Fernando Silva-Coira. Clustering-based compression for raster time series. *The Computer Journal*, 68(1):32–46, 09 2024.

[119] Martita Muñoz, José Fuentes-Sepúlveda, Cecilia Hernández, and Diego Seco. Estimating the compressibility of raster data. *Information Systems*, 136:102624, 2026.

[120] Gonzalo Navarro. Wavelet trees for all. *Journal of Discrete Algorithms*, 25:2–20, 2014. 23rd Annual Symposium on Combinatorial Pattern Matching.

[121] Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, Cambridge, England, 2016.

[122] Gonzalo Navarro. Indexing highly repetitive string collections, part i: Repetitiveness measures. *ACM Comput. Surv.*, 54(2), mar 2021.

[123] Gonzalo Navarro, Carlos Ochoa, and Nicola Prezza. On the approximation ratio of ordered parsings. *IEEE Transactions on Information Theory*, 67(2):1008–1026, 2021.

[124] Gonzalo Navarro and Eliana Providel. Fast, small, simple rank/select on bitmaps. In *SEA*, pages 295–306, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[125] Gonzalo Navarro and Cristian Urbina. On stricter reachable repetitiveness measures. In *SPIRE*, pages 193–206, Cham, 2021. Springer International Publishing.

[126] Gonzalo Navarro and Cristian Urbina. Repetitiveness measures based on string morphisms. *Theoretical Computer Science*, 1043:115259, 2025.

[127] Takaaki Nishimoto, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda, et al. Fully dynamic data structure for lce queries in compressed space. In *MFCS*, page 72:1–72:15, 2016.

[128] Takaaki Nishimoto and Yasuo Tabei. Lzrr: Lz77 parsing with right reference. *Information and Computation*, 285:104859, 2022.

[129] CUDA Nvidia. Parallel programming and computing platform, 2014.

[130] Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *ALENEX*, pages 60–70. SIAM, 2007.

[131] Daisuke Okanohara and Kunihiko Sadakane. A linear-time burrows-wheeler transform using induced sorting. In *SPIRE*, pages 90–101, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[132] J Keith Ord and Arthur Getis. Local spatial autocorrelation statistics: distributional issues and an application. *Geographical analysis*, 27(4):286–306, 1995.

[133] Rasmus Pagh. Low redundancy in static dictionaries with o(1) worst case lookup time. In *ICALP*, pages 595–604, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[134] José M Pena, Jose Antonio Lozano, and Pedro Larranaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern recognition letters*, 20(10):1027–1040, 1999.

[135] W. D. Pence, R. Seaman, and R. L. White. Lossless astronomical image compression and the effects of noise. *Publications of the Astronomical Society of the Pacific*, 121(878):414, may 2009.

[136] A. Pinto, D. Seco, and G. Gutiérrez. Improved queryable representations of rasters. In *DCC*, pages 320–329, 2017.

[137] Md Atiqur Rahman, Mohamed Hamada, and Jungpil Shin. The impact of state-of-the-art techniques for lossless still image compression. *Electronics*, 10(3):360, 2021.

[138] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43–es, November 2007.

[139] Lior Rokach and Oded Maimon. *Clustering Methods*, pages 321–352. Springer US, Boston, MA, 2005.

[140] Giuseppe Romana, Marinella Sciortino, and Cristian Urbina. Exploring repetitiveness measures for two-dimensional strings. *arXiv preprint arXiv:2404.07030*, 2024.

[141] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[142] Luís M. S. Russo, Ana D. Correia, Gonzalo Navarro, and Alexandre P. Francisco. Approximating optimal bidirectional macro schemes. In *DCC*, pages 153–162, 2020.

[143] Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory of Computing Systems*, 41(4):589–607, 2007.

[144] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, June 1984.

[145] K.N. Satone, A.S. Deshmukh, and P.B. Ulhe. A review of image compression techniques. In *ICECA*, volume 1, pages 97–101, 2017.

[146] Shi Shan. On the generalized zipf distribution. part i. *Information Processing & Management*, 41(6):1369–1386, 2005. Special Issue on Infometrics.

[147] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[148] Julian Shun. Fast parallel computation of longest common prefixes. In *SC*, pages 387–398, 2014.

[149] Julian Shun. Parallel wavelet tree construction. In *DCC*, pages 63–72, 2015.

[150] Julian Shun. Improved parallel construction of wavelet trees and rank/select structures. *Information and Computation*, 273:104516, 2020. DCC (Data Compression Conference) 2018.

[151] Julian Shun and Fuyao Zhao. Practical parallel lempel-ziv factorization. In *DCC*, pages 123–132, 2013.

[152] Fernando Silva-Coira, José R Paramá, and Susana Ladra. Map algebra on raster datasets represented by compact data structures. *Software: Practice and Experience*, 53(6):1362–1390, 2023.

[153] Fernando Silva-Coira, José R. Paramá, Guillermo Bernardo, and Diego Seco. Space-efficient representations of raster time series. *Information Sciences*, 2021.

[154] Fernando Silva-Coira, José R. Paramá, Susana Ladra, Juan R. López, and Gilberto Gutiérrez. Efficient processing of raster and vector data. *PLOS ONE*, 15(1):1–35, 01 2020.

[155] Neha Sisodiya, Sanjay Garg, Nitant Dube, Priyank Thakkar, Akshay Parmar, and Shashikant Sharma. Scalable clustering for eo data using efficient raster representation. *Multimedia Tools and Applications*, 82(8):12303–12319, 2023.

[156] Shashank Srivastav, Pradeep Kumar Singh, and Divakar Yadav. A method to improve exact matching results in compressed text using parallel wavelet tree. *Scalable Computing: Practice and Experience*, 22(4):387–400, 2021.

[157] Artur Starczewski and Adam Krzyżak. Performance evaluation of the silhouette index. In *ICAISC*, pages 49–58. Springer,Cham, 6 2015.

[158] Douglas Steinley. K-means clustering: a half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59(1):1–34, 2006.

[159] James A Storer and Thomas G Szymanski. Data compression via textual substitution. *Journal of the ACM (JACM)*, 29(4):928–951, 1982.

[160] MTCAJ Thomas and A Thomas Joy. *Elements of information theory*. Wiley-Interscience, 2006.

[161] Waldo R Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1):234–240, 1970.

[162] Antony Unwin. Geary's contiguity ratio. *The Economic and Social Review*, 27(2):145–159, 1996.

[163] Eli Veda Sai Rochishna, Vutukuri Ganesh Hari Prasad Rao, Gandhavarapu Bhargav, and S. Sathyalakshmi. Lossless image compression using machine learning. In *ICSADL*, pages 113–125, Singapore, 1 2023. Springer Nature Singapore.

[164] Leyuan Wang, Sean Baxter, and John D. Owens. Fast parallel suffix array on the gpu. In *Euro-Par*, pages 573–587, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[165] Xiaozhu Wu and Ximei Zhang. An efficient pixel clustering-based method for mining spatial sequential patterns from serial remote sensing images. *Computers & Geosciences*, 124:128–139, 2019.

[166] Y Xia, K Mitchell, M Ek, J Sheffield, B Cosgrove, E Wood, L Luo, C Alonge, H Wei, J Meng, et al. Nldas primary fosrcing data l4 hourly 0.125$\times$ 0.125 degree v002. goddard earth sciences data and information services center (ges disc), greenbelt, md, usa, rep. Technical report, NASA/GSFC/HSL, Maryland, Washington DC, USA, 2009.

[167] Arun Kumar Yadav, Divakar Yadav, Akhilesh Verma, Mohd Akbar, and Kartikey Tewari. Scalable thread based index construction using wavelet tree. *Multimedia Tools and Applications*, 82(9):14037–14053, 2023.

[168] Hai Jing Zhu, Bo Chong Han, and Bo Qiu. Survey of astronomical image processing methods. In *ICIG*, pages 420–429, Cham, 2015. Springer International Publishing.

[169] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.