



Universitat Autònoma de Barcelona

**ADVERTIMENT.** L'accés als continguts d'aquesta tesi queda condicionat a l'acceptació de les condicions d'ús establertes per la següent llicència Creative Commons:  [http://cat.creativecommons.org/?page\\_id=184](http://cat.creativecommons.org/?page_id=184)

**ADVERTENCIA.** El acceso a los contenidos de esta tesis queda condicionado a la aceptación de las condiciones de uso establecidas por la siguiente licencia Creative Commons:  <http://es.creativecommons.org/blog/licencias/>

**WARNING.** The access to the contents of this doctoral thesis it is limited to the acceptance of the use conditions set by the following Creative Commons license:  <https://creativecommons.org/licenses/?lang=en>



Universitat Autònoma de Barcelona

Departament d'Enginyeria de la Informació i  
de les Comunicacions

**COMPACT DATA STRUCTURES  
FOR  
REMOTE SENSING DATA**

SUBMITTED TO UNIVERSITAT AUTÒNOMA DE BARCELONA  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  
DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

by Kevin Chow  
Bellaterra, July 2022

Supervisors:  
Dr. Joan Serra-Sagristà  
Dr. Ian Blanes

© Copyright 2022 by Kevin Chow

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Bellaterra, July 2022

---

Dr. Joan Serra-Sagristà

Dr. Ian Blanes

*Committee:*

Dr. Joan Bartrina-Rapesta

Dr. Fernando Silva Coira

Dr. Cecilia Hernández Rivas

(substitute) Dr. Guillermo Navarro

(substitute) Dr. Nieves Rodríguez Brisaboa

(substitute) Dr. José Fuentes-Sepúlveda







# Abstract

In this digital era, an enormous amount of data are being generated and processed daily. They accumulate to such an extent that it is necessary and imperative to use data compression to reduce the data size so that they can take up as little space as possible. Among the many data compression schemes, there is one known as compact data structures, and it will be the focus of this thesis. These structures store data efficiently while also providing real-time access to the data in the compressed domain, i.e., to query an individual element, it is not necessary to decompress the whole structure. Compact data structures also provide lossless compression, thus ensuring no information loss during the compression process.

Remote sensing hyperspectral scenes are image data that are transmitted from sensors located in aircraft or in satellites orbiting the Earth to receivers at ground stations. Due to the size of the data, they need to be compressed in such a way that they can be transmitted more quickly and when they reach the ground stations, they can be stored in an efficient manner to save space. Therefore, data compression is necessary for faster transmission and reduced storage space.

This thesis sets out to explore several distinct ways of using compact data structures to provide better performance with regard to compression ratio and access time for remote sensing hyperspectral data. First, we describe a predictive method and a differential method designed to work with a compact data structure and evaluate the improvements made. Then we present a study of different variable-length codes that can be used in tandem with compact data structures to achieve higher compression gains. Next, we analyze the tree structure of the raster matrix so that only nodes that contain relevant data are saved, thus making the structure more compact. Finally, we investigate a recently proposed compact data structure and examine how its performance stacks up against the others.

Experiments have shown that these proposed methods produce results that remain competitive with the traditional techniques and methods that have been in use.





# Acknowledgements

First, let me express my utmost gratitude to Dr. Joan Serra-Sagristà and Dr. Ian Blanes for their patient and tireless guidance and for always providing me with their invaluable inputs and helpful advice. Their support has made this Ph.D. dissertation possible and an eventual reality, and I appreciate very much their time and commitment.

I am immensely grateful to Dr. Diego Seco for giving me the opportunity to collaborate with him during my virtual research stay. His insights into the subject of compact data structures will be relished forever.

The love and support afforded me by my dearest parents and Lena, my favorite and only sister, just goes to show that there are things in life that you cannot put a price tag on.

Thanks to all my friends and relatives around the world in Hong Kong, Spain, Canada, China, and Taiwan. Special thanks go to Wah, Sin Chuk, Lim, Jo, Cousin Chiu, and Kwok Shuen.

Finally, thanks go to all those who work in my department, DEIC, for making my research endeavor and my teaching job a little less hectic and a more enjoyable experience.



# Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Compact data structures . . . . .	1
1.1.1 $K^2$ -tree structure . . . . .	2
1.1.2 $K^2$ -raster structure . . . . .	2
1.1.3 T- $k^2$ raster structure . . . . .	5
1.1.4 Rank and select functions . . . . .	5
1.2 Remote sensing data . . . . .	7
1.3 Variable-length encoding . . . . .	8
1.4 Contributions to the thesis . . . . .	9
1.5 Organization of the thesis . . . . .	10
<b>2 Using predictive and differential methods with <math>k^2</math>-raster compact data structure for hyperspectral image lossless compression</b>	<b>13</b>
<b>3 Analysis of variable-length codes for integer encoding in hyperspectral data compression with the <math>k^2</math>-raster compact data structure</b>	<b>39</b>
<b>4 Performance improvement on <math>k^2</math>-raster compact data structure for hyperspectral scenes</b>	<b>57</b>

<b>5</b>	<b>A compact data structure for hyperspectral scenes based on raster time series</b>	<b>63</b>
<b>6</b>	<b>Results summary</b>	<b>69</b>
6.1	Use of different $k$ -values . . . . .	69
6.1.1	Storage size . . . . .	71
6.1.2	Access time . . . . .	71
6.2	Predictive and differential methods . . . . .	72
6.2.1	Storage size . . . . .	73
6.2.2	Access time . . . . .	73
6.2.3	Group size . . . . .	74
6.3	Integer encoders . . . . .	74
6.3.1	Storage size . . . . .	74
6.3.2	Access time . . . . .	75
6.4	Padding versus unpadding . . . . .	75
6.4.1	Storage size . . . . .	75
6.4.2	Access time . . . . .	76
6.5	T- $k^2$ raster structure . . . . .	76
6.5.1	Storage size . . . . .	77
6.5.2	Access time . . . . .	78
<b>7</b>	<b>Conclusion</b>	<b>79</b>
7.1	Summary . . . . .	80
7.2	Future work . . . . .	81

# Chapter 1

## Introduction

### 1.1 Compact data structures

Compact data structures [1] are losslessly compressed structures that provide efficient storage and random data access. They have evolved from the 1990s after the work of Guy Jacobson [2, 3] who proposed the use of the rank and select functions as their main primitive operations to locate the elements of a structure. This makes it possible to avoid using pointers, which usually occupy a lot of space in the structure.

The second advantage is the structure's ability to offer random access to individual elements without full decompression. In that respect, it stands apart from data compressed by other popular compression techniques such as Gzip or specialized algorithms such as CCSDS 123.0-B-2 [4].

The third advantage is that the smaller size of the structure allows it to fit into memory and cache, making it less likely to be swapped out to disk, resulting in faster queries. Compact data structures also provide lossless compression which means the original data can be restored and rebuilt from the compressed data and the quality stays the same.

In this thesis, several compact data structures are studied and investigated including  $k^2$ -tree [5],  $k^2$ -raster [6], and T- $k^2$ -raster [7]. They will be discussed in greater detail in Chapter 2 through Chapter 5, but a brief description of these structures and some related information are given below.

### 1.1.1 $K^2$ -tree structure

$K^2$ -tree was originally proposed to be used in Web graphs, social networks, etc. It is built from a binary adjacency matrix (with values 0 or 1) based on graphs. Fig. 1.1 illustrates how a  $k^2$ -tree is built from an  $8 \times 8$  binary adjacency matrix which corresponds to a graph with eight nodes.

$K^2$ -tree is built by recursively partitioning the matrix into square submatrices of equal size until each submatrix reaches a size of  $k \times k$  and  $k \geq 2$ . When the matrix is being partitioned into submatrices, if at least one cell in the submatrix is found to have a value of 1, then the node of the tree will be set to 1. Otherwise, the node will be set to 0, which means that it is now a leaf node and does not have any children. In this case, the partitioning stops in this submatrix and will not go any further.

### 1.1.2 $K^2$ -raster structure

$K^2$ -raster (or  $k^2$ raster) was proposed based on the work done in  $k^2$ -tree. The tree structure is built from a matrix with integer values. Therefore, the nodes of the tree also contain integers. Fig. 1.2 shows how a  $k^2$ -raster is built from an  $8 \times 8$  matrix and how the values are stored in the tree nodes.

$K^2$ -raster consists of several basic elements: a bitmap ( $T$ ), two integer arrays ( $L_{max}$ ,  $L_{min}$ ), a variable-length encoder, and a  $k^2$ -ary tree. The raster matrix is recursively partitioned until each submatrix reaches a size of  $k \times k$ . However, if all the elements in the submatrix have the same value, the partitioning will stop, and the corresponding tree node becomes a leaf node without any children. A  $T$  bitmap which is generated from all the nodes at all levels except for the root and the last level indicates which nodes contain child nodes (1) and which ones do not (0). This bitmap is used by the rank function in query to find the element among the tree nodes.

Unlike  $k^2$ -tree, at each tree level, the maximum and minimum values of each

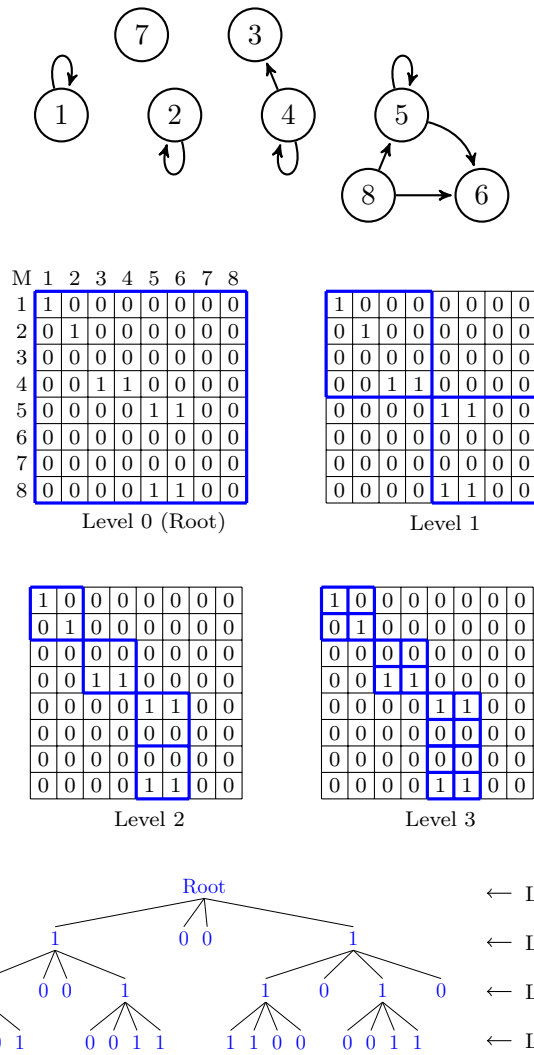


Figure 1.1: (Top) A graph with 8 nodes. (Middle) The corresponding  $8 \times 8$  binary adjacency matrix at various stages of recursive partitioning. (Bottom) A  $k^2$ -tree ( $k = 2$ ) constructed from the matrix.



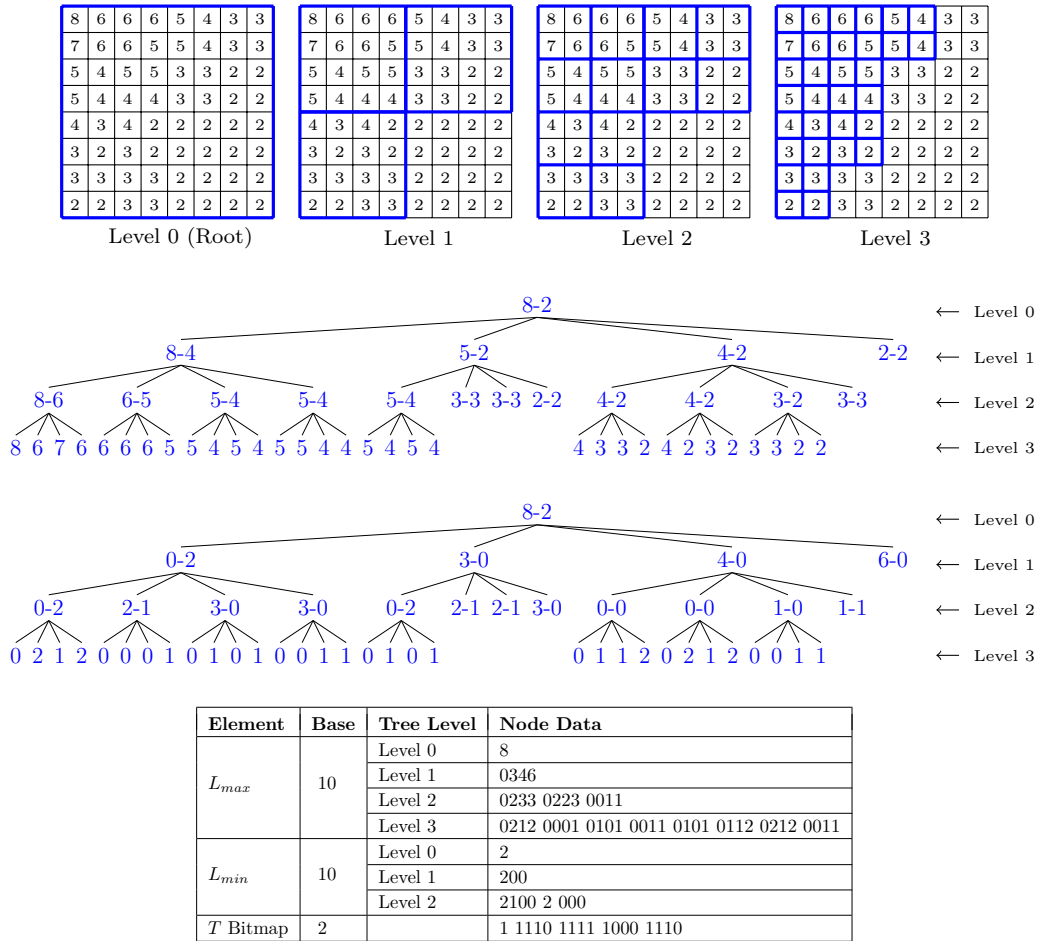


Figure 1.2: (Top) An  $8 \times 8$  matrix ( $M_s$ ) showing recursive partitioning. (Middle) The upper tree is a  $k^2$ -raster ( $k = 2$ ) tree constructed from the matrix and the lower tree takes into account the differences between the parent and child nodes. Except for the last level, maximum and minimum values in each node are separated by a hyphen. (Bottom) A table showing the elements of the  $k^2$ -raster.

submatrix are stored in each tree node and these values go into two arrays which are called  $V_{max}$  and  $V_{min}$  respectively. To further improve compression, at each tree node, the differences of the maximum and minimum values between the parent and child nodes are taken and they will replace the values stored in the child nodes. With smaller values in each node, a better compression can be achieved with the variable-length encoder.

The root's maximum ( $rMax$ ) and minimum ( $rMin$ ) are integer values that remain uncompressed and become the first element of an  $L_{max}$  and an  $L_{min}$  array respectively. The  $V_{max}$  arrays at the other levels are then concatenated to the  $L_{max}$  array while the  $V_{min}$  arrays at the other levels (except the last level) are concatenated to the  $L_{min}$  array.

### 1.1.3 T- $k^2$ raster structure

T- $k^2$ raster is for 3D rasters such as hyperspectral data where the third dimension specifies the range of wavelengths or the spectral band or component. It is composed of two kinds of  $k^2$ -rasters: a *snapshot* matrix and a *log* matrix. The snapshot matrix is a normal  $k^2$ -raster while the log matrix contains the differences taken between the corresponding elements in this matrix and the ones in the snapshot matrix. However, we should note that a bit unlike a normal  $k^2$ -raster, when the log matrix is in the process of being partitioned recursively, the partitioning of the submatrix will stop if the differences of all the elements in the submatrix are the same (zero or otherwise) or the values of all the elements in the submatrix are the same. Fig. 1.3 shows how a T- $k^2$ raster is built from an  $8 \times 8$  matrix and how, after taking the differences between this log matrix ( $M_{s+1}$ ) and the snapshot matrix in Figure 1.2 ( $M_s$ ), the values are stored in the tree nodes.

### 1.1.4 Rank and select functions

In compact data structures, there are many uses of the rank and select functions. For example, in  $k^2$ -raster, the rank function is used for finding the location of the nodes in the tree structure. Section 2.2 "LOUDS" in Chapter 2 will provide more details

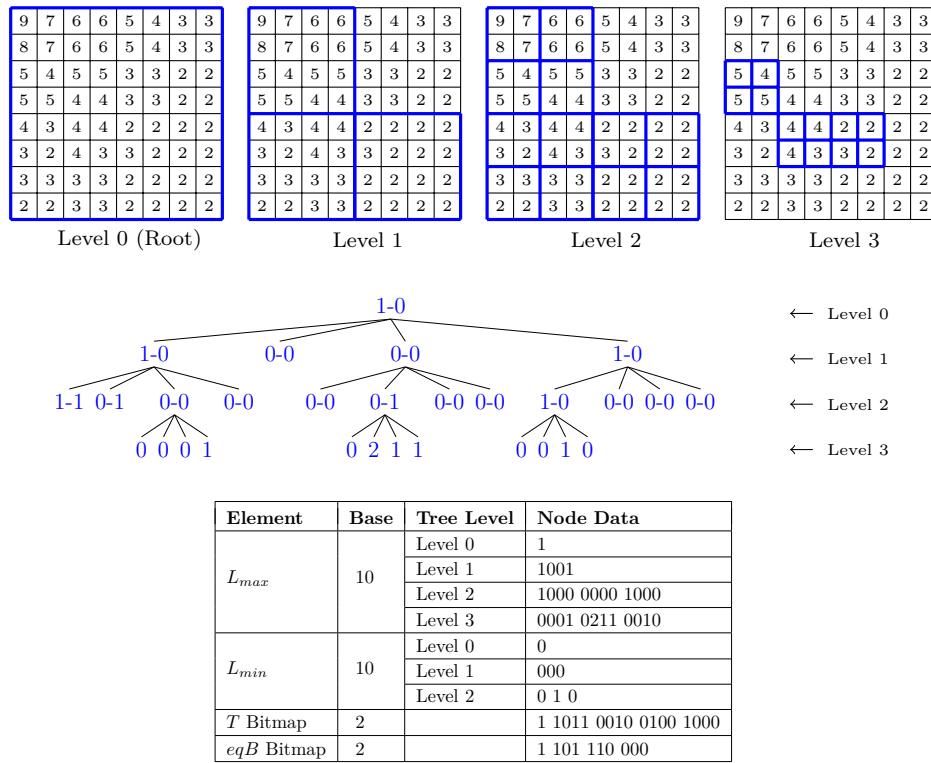


Figure 1.3: (Top) A non-snapshot matrix ( $M_{s+1}$ ) showing recursive partitioning. (Middle) a  $T-k^2$  raster ( $k = 2$ ) tree constructed from the differences between the matrix ( $M_{s+1}$ ) at the top of this figure and the matrix in Fig. 1.2 ( $M_s$ ). Except for the last level, all nodes show the maximum and minimum values separated by a hyphen. (Bottom) A table showing the elements of the  $T-k^2$  raster.

on these functions.

$\text{rank}_b(m)$  returns the number of bits with a value of  $b$  starting from the first position up to and including the position  $m$  in the bitmap ( $b$  is 0 or 1).

$\text{select}_b(i)$  returns the position of the  $i$ -th bit that has a value of  $b$  in the bitmap ( $b$  is 0 or 1).

By default,  $b$  is 1, i.e.,  $\text{rank}(m) = \text{rank}_1(m)$ . These operations are inverses of each other, i.e.,  $\text{rank}_b(\text{select}_b(m)) = \text{select}_b(\text{rank}_b(m)) = m$ .

## 1.2 Remote sensing data

Remote sensing data refer to the information received from satellites in outer space or from aircraft. These data are obtained by measuring the reflected and emitted radiation of a certain area on Earth and are composed of multiple bands spanning across the electromagnetic spectrum. The data in a range of bands in this spectrum help us look for objects that we are interested in, for example, minerals [8], oil fields [9], and agriculture [10]. They also help us with applications such as wildfire soil studies [11] and weather prediction [12]. Hyperspectral scenes, which are a type of remote sensing data, are the focus of this thesis. They are captured by sensors in real missions from aircraft such as Airborne Visible/Infrared Imaging Spectrometer (AVIRIS), or from satellites such as Atmospheric Infrared Sounder (AIRS), Compact Reconnaissance Imaging Spectrometer for Mars (CRISM), Hyperion, and Infrared Atmospheric Sounding Interferometer (IASI). These sensors are instruments from the National Aeronautics and Space Administration (NASA) except for IASI which is from the European Space Agency (ESA).

It should be noted that hyperspectral data are 3D data. For example, given the sizes  $x$ ,  $y$  and  $z$ ,  $x$  and  $y$  are the width and height of the raster matrix respectively, and  $z$  is the spectral range of wavelengths. When compressing hyperspectral data,

we can take advantage of two aspects of the data: spatial and spectral correlations. Spatial correlation is related to the similarities or redundancies of the data within the raster matrix whereas spectral correlation is related to the similarities or redundancies of the corresponding elements in two different spectral bands. In general, for spectral correlation, the closer two spectral bands are to each other, the higher the redundancies are.

### 1.3 Variable-length encoding

In order to compress a compact data structure, variable-length encoders are needed. These encoders allow compact data structures to compress the data so that access to the individual elements can be accomplished with a minimal degree of decompression. In the original paper on  $k^2$ -raster, a variable-length encoder or integer encoder known as Directly Addressable Codes (DACs) [13] was used to compress an array of data from the tree nodes in the structure so that the resulting data occupy less space. However, in our research, other integer encoders have also shown similar or even more competitive results when compared with DACs. We have explored a few of them, including Rice codes [14], Simple-9 [15], Simple-16 [16], and PForDelta codes [17], among others, and they are briefly described below beginning with the simple unary codes:

- Unary codes are bit-aligned codes for small integers. If  $x$  is a non-negative integer and  $|x|$  is the minimum bit length to express  $x$  ( $|x| = \lfloor \log_2 x \rfloor + 1$ ), then unary codes are defined as:

$$u(x) = 0^x 1 , \quad (1.1)$$

where the superscript  $x$  indicates the number of consecutive 0 bits in the code.

- Rice codes are bit-aligned codes. If  $x$  is an integer value in the sequence and  $y = \lfloor x/2^l \rfloor$ , with  $l$  being a non-negative integer parameter, the Rice codes for this parameter are defined as:

$$R_l(x) = u(y + 1) [x]_l . \quad (1.2)$$

In Section 2.4 “Rice Codes” of Chapter 3, a more detailed discussion of Rice codes will be presented.

- DACs are a variable-length encoding scheme for direct access to encoded integer sequences. It does not require a sampling function. In Section 2.6 “Directly Addressable Codes” of Chapter 3, a more detailed discussion of DACs will be presented.
- Simple-9 is a word-aligned encoding scheme where a 32-bit word is split into two parts: a 28-bit part containing a variable number of integers being encoded and a 4-bit part which is a selector with a value ranging from 0 to 8.
- Simple-16 is similar to Simple-9. It uses all the 16 combinations in the selector with values ranging from 0 to 15.
- PForDelta is another word-aligned encoding scheme. It encodes a fixed group of 32, 64, 128 or 256 integers so that they can fit into a certain number of bytes. A percentage of those integers that are larger than the others are encoded separately and are placed after the smaller integers or in another location.

For Simple-9, Simple-16 and PForDelta, please refer to Section 2.5 in Chapter 3 for a more comprehensive review of these codes.

## 1.4 Contributions to the thesis

- Kevin Chow, Dion Eustathios Olivier Tzamarias, Ian Blanes, and Joan Serra-Sagristà, “**Using Predictive and Differential Methods with  $k^2$ -raster Compact Data Structure for Hyperspectral Image Lossless Compression,**” *MDPI Remote Sensing*, vol. 11, no. 11, 2019.  
DOI: 10.3390/rs11212461. [18] (IF: 4.509, Q2)
- 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  $k^2$ -raster Compact Data Structure,” *MDPI Remote Sensing*, vol. 12, no. 12, 2020. DOI: 10.3390/rs12121983. [19] (IF: 4.848, Q1)

- Kevin Chow, Dion Eustathios Olivier Tzamaras, Miguel Hernández-Cabronero, Ian Blanes, and Joan Serra-Sagristà, “Performance Improvement on  $k^2$ -raster Compact Data Structure for Hyperspectral Scenes,” *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2021. DOI: 10.1109/LGRS.2021.3084065. [20] (IF: 3.966, Q1)
- Kevin Chow, Dion Eustathios Olivier Tzamaras, Miguel Hernández-Cabronero, Ian Blanes, and Joan Serra-Sagristà, “A Compact Data Structure for Hyperspectral Scenes Based on Raster Time Series,” SUBMITTED in April 2022 to *IEEE Geoscience and Remote Sensing Letters*. (IF: 3.966, Q1)

## 1.5 Organization of the thesis

The rest of the thesis is organized as follows:

**Chapter 2** presents our publication [18] where a lossless coder is proposed for compression and real-time processing of hyperspectral data. A predictor or a differential encoder is applied to a collection of raster matrices to reduce their bit rates by exploiting the similarities in pixel values between corresponding elements in neighboring bands. The encoder then uses  $k^2$ -raster to further reduce the bit rates and attains a better compression ratio.

**Chapter 3** presents our paper [19] in which we study a number of variable-length encoders that encode integer data in a  $k^2$ -raster for hyperspectral scenes. For a competitive performance, choosing the right integer encoder is important for the com-

pression ratio and access time. In this chapter various integer encoders are discussed including Rice, Simple-9, Simple-16, PForDelta codes, and DACs. This chapter also describes a heuristic method to compute an optimal  $k$ -value that can be used to build a  $k^2$ -raster that produces the best performance.

**Chapter 4** presents our publication [20] where we continue to explore ways of improving data size and access time for  $k^2$ -raster. We study the raster matrix without any padding, i.e., an unpadding matrix and determine whether we can still compress the structure and access the data, and how it fares compared with a padded matrix. Next, we examine some integer encoders that are word-aligned codes, specifically the Simple family and PForDelta and discuss how well their random element access is. A comparison with that of DACs is presented.

**Chapter 5** presents our latest manuscript which was submitted to IEEE Geoscience and Remote Sensing Letters (GRSL) journal in April 2022. It examines T- $k^2$ -raster, a compact data structure that was recently proposed based on  $k^2$ -raster. This structure is a combination of  $k^2$ -raster and a modified version of  $k^2$ -raster, both of which are  $k^2$ -ary tree data structures that provide efficient storage and real-time processing. T- $k^2$ -raster, which in earlier research produced favorable results for raster time series, is now being studied for remote sensing data. The structure can be used to minimize redundancies in hyperspectral data resulting from the spectral correlation between elements in adjacent bands.

**Chapter 6** presents our analysis of the results from experiments that have been performed by the different methods described in chapter 2 through 5 using compact data structures.

**Chapter 7** concludes this thesis with a summary as well as several interesting ideas for projects that we can pursue in the future.





## Chapter 2

Using predictive and differential methods with  $k^2$ -raster compact data structure for hyperspectral image lossless compression



Article

# Using Predictive and Differential Methods with $K^2$ -Raster Compact Data Structure for Hyperspectral Image Lossless Compression <sup>†</sup>

Kevin Chow \*, Dion Eustathios Olivier Tzamarías, Ian Blanes and Joan Serra-Sagristà

Department of Information and Communications Engineering, Universitat Autònoma de Barcelona, 08193 Cerdanyola del Vallès, Barcelona, Spain; dion.tzamarías@uab.cat (D.E.O.T.); ian.blanes@uab.cat (I.B.); joan.serra@uab.cat (J.S.-S.)

\* Correspondence: kevin.chow@uab.cat

<sup>†</sup> This paper is an extended version of our paper published in the 6th ESA/CNES International Workshop on On-Board Payload Data Compression Proceedings.

Received: 31 August 2019; Accepted: 17 October 2019; Published: 23 October 2019



**Abstract:** This paper proposes a lossless coder for real-time processing and compression of hyperspectral images. After applying either a predictor or a differential encoder to reduce the bit rate of an image by exploiting the close similarity in pixels between neighboring bands, it uses a compact data structure called  $k^2$ -raster to further reduce the bit rate. The advantage of using such a data structure is its compactness, with a size that is comparable to that produced by some classical compression algorithms and yet still providing direct access to its content for query without any need for full decompression. Experiments show that using  $k^2$ -raster alone already achieves much lower rates (up to 55% reduction), and with preprocessing, the rates are further reduced up to 64%. Finally, we provide experimental results that show that the predictor is able to produce higher rates reduction than differential encoding.

**Keywords:** compact data structure; quadtree;  $k^2$ -tree;  $k^2$ -raster; DACs; 3D-CALIC; M-CALIC; hyperspectral images

## 1. Introduction

Compact data structures [1] are examined in this paper as they can provide real-time processing and compression of remote sensing images. These structures are stored in reduced space in a compact form. Functions can be used to access and query each datum or groups of data directly in an efficient manner without an initial full decompression. This compact data should also have a size which is close to the information-theoretic minimum. The idea was explored and examined by Guy Jacobson in his doctoral thesis in 1988 [2] and in a paper published by him a year later [3]. Prior to this, works had been done to express similar ideas. However, Jacobson's paper is often considered the starting point of this topic. Since then it has gained more attention and a number of research papers have been published. Research on algorithms such as FM-index [4,5] and Burrows-Wheeler transform [6] were proposed and applications were released, notable examples of which include bzip2 (<https://linux.die.net/man/1/bzip2>), Bowtie [7] and SOAP2 [8]. One of the advantages of using compact data structures is that the compressed data form can be loaded into main memory and accessed directly. The smaller compressed size also helps data move through communication channels faster. The other advantage is that there is no need to compress and decompress the data as is the case with data compressed by a classical compression algorithm such as gzip or bzip2, or by a specialized algorithm such as CCSDS 123.0-B-1 [9] or KLT+JPEG 2000 [10,11]. The resulting image will have the same quality as the original.

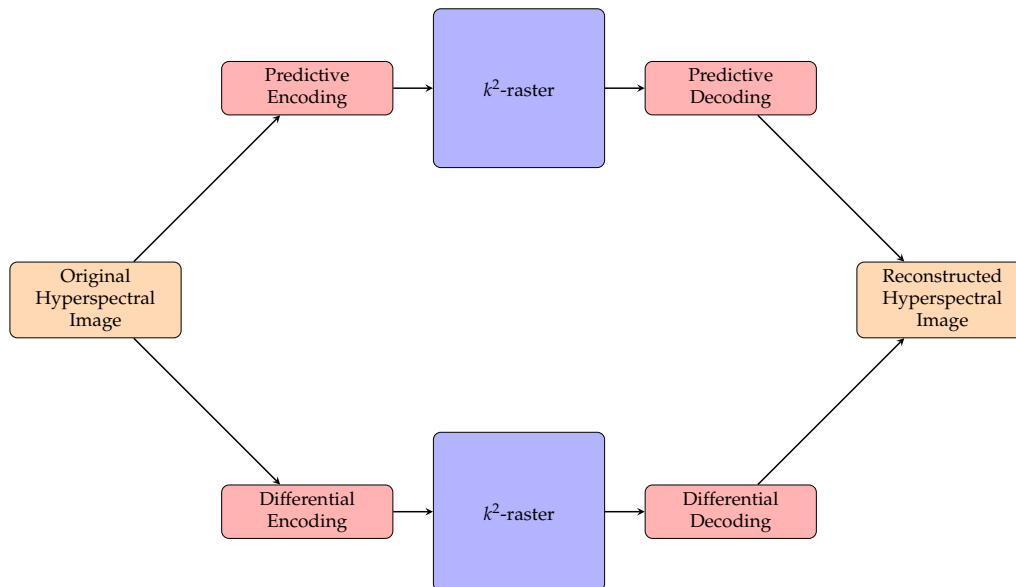
Hyperspectral images are image data that contain a multiple number of bands from across the electromagnetic spectrum. They are usually taken by hyperspectral satellite and airborne sensors. Data are extracted from certain bands in the spectrum to help us find the objects that we are specifically looking for, such as oil fields and minerals. However, due to their large sizes and the huge amount of data that have been collected, hyperspectral images are normally compressed by lossy and lossless algorithms to save space. In the past several decades, a lot of research studies have gone into keeping the storage sizes to a minimum. However, to retrieve the data, it is still necessary to decompress all the data. With our approach using compact data structures, we can query the data without fully decompressing them in the first place, and this is the main motivation for this work.

Prediction is one of the schemes used in lossless compression. CALIC (Context Adaptive Lossless Image Compression) [12,13] and 3D-CALIC [14] belong to this class of scheme. In 1994, Wu et al. introduced CALIC, which uses both context and prediction of the pixel values. In 2000, the same authors proposed a related scheme called 3D-CALIC in which the predictor was extended to the pixels between bands. Later in 2004, Magli et al. [15] proposed M-CALIC whose algorithm is related to 3D-CALIC. All these methods take advantage of the fact that in a hyperspectral image, neighboring pixels in the same band (spatial correlation) are usually close to each other and even more so for neighboring pixels of two neighboring bands (spectral correlation).

Differential encoding is another way of encoding an image by taking the difference between neighboring pixels and in this work, it is a special case of the predictive method. It only takes advantage of the spectral correlation. However, this correlation between the pixels in the bands will become smaller as the distance between the bands are further apart and therefore, its effectiveness is expected to decrease when the bands are far from each other.

The latest studies on hyperspectral image compression, both lossy and lossless, are focused on CCSDS 123.0, vector quantization, Principal Component Analysis (PCA), JPEG2000, and Lossy Compression Algorithm for Hyperspectral Image Systems (HyperLCA), among many others. Some of these research works are listed in [16–19]. In this work, however, we investigate lossless compression of hyperspectral images through the proposed  $k^2$ -raster for 3D images, which is a compact data structure that can provide bit-rate reduction as well as direct access to the data without full decompression. We also explore the use of a predictor and a differential encoder as preprocessing on the compact data structure to see if it can provide us with further bit-rate reduction. The predictive method and the differential method are also compared. The flow chart shown in Figure 1 depicts how the encoding/decoding of this proposal works.

This paper is organized as follows: In Section 2, we present the  $k^2$ -raster and discuss it in detail, beginning with quadtree, followed by  $k^2$ -tree and  $k^2$ -raster. Later in the same section, details of the predictive method and the differential method are discussed. Section 3 shows the experimental results on how the two methods fare using  $k^2$ -raster on hyperspectral images, and more results on how some other factors such as using different  $k$ -values can affect the bit rates. Finally, we present our conclusions in Section 4.



**Figure 1.** A flow chart showing the encoding and decoding of this coder.

## 2. Materials and Methods

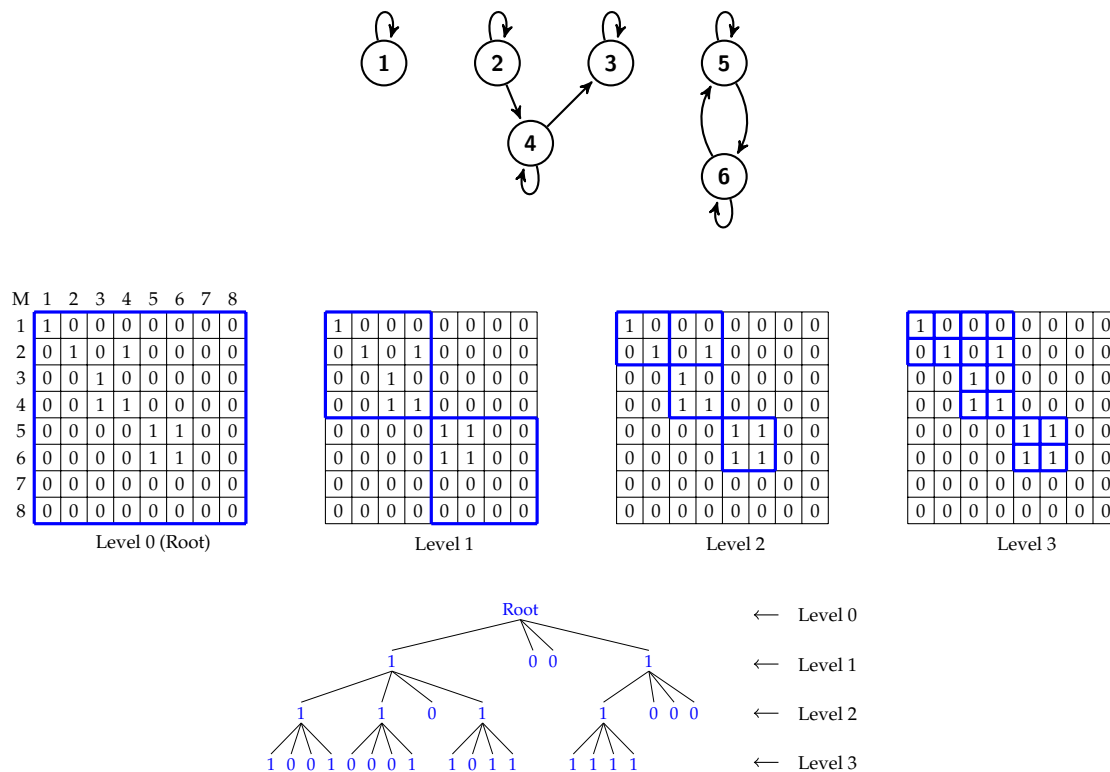
One way to build a structure that is small and compact is to make use of a tree structure and do it without using pointers. Pointers usually take up a large amount of space, with each one having a size in the order of 32 or 64 bits for most modern-day machines or programs. A tree structure with  $n$  pointers will have a storage complexity of  $\mathcal{O}(n \log n)$  whereas a pointer-less tree only occupies  $\mathcal{O}(n)$ . For pointer-less trees, to get at the elements of the structure, rank and select functions [3] are used, and that only requires simple arithmetic to find the parent's and child's positions. This is the premise that compact data structures are based on. In this work, we will use  $k^2$ -raster from Ladra et al. [20], a concept which was developed from  $k^2$ -tree, also a type of compact data structure, as well as the idea of using recursive decomposition of quadtrees. The results of  $k^2$ -raster were quite favorable for the data sets that were used. Therefore, we are extending their approach for hyperspectral images and investigate whether it would be possible to use that structure for 3D hyperspectral images. The Results section will show us that the results are quite competitive compared to other commonly-used classical compression techniques. There is a bit-rate reduction of up to 55% for the testing images. Upon more experimentation with predictive and differential preprocessing, a further bit-rate reduction of up to 64% can be attained. For that reason, we are proposing in this paper our encoder using the predictor or differential method on  $k^2$ -raster for hyperspectral images.

### 2.1. Quadtrees

Quadtree structures [21], which have been used in many kinds of data representations such as image processing and computer graphics, are based on the principle of recursive decomposition. As there are many variants of quadtree, we will describe the one that is pertinent to our discussion: region quadtree. Basically, a quadtree is a tree structure where each internal node has 4 children. Given a 2D square matrix, it is partitioned recursively into four equal subquadrants. If a tree is built to represent this, it will have a root node at level 0 with 4 children nodes at level 1, each child representing a node and a subquadrant. Next, if the subquadrant has a size larger than  $2^2$ , then each of these subquadrants will be partitioned to give 4 more children and a new level 2 is added to the tree. Note that the tree nodes are traversed in a left to right order.

Considering a matrix of size  $n \times n$  where  $n$  is a power of 2, it is recursively divided until each subquadrant has a size of  $2^2$ . For example, if the size of the matrix is  $8 \times 8$ , after the recursive division of matrix,  $(8^2)/(2^2) = 16$  subquadrants are obtained. It should be noted that the value of  $n$  in the image matrix needs to be a power of 2. Otherwise, the matrix has to be enlarged widthwise and heightwise to

a value which is the next power of 2, and these additional pixels will be padded with zeros. As  $k^2$ -trees are based on quadtrees, the division and the resulting tree of a quadtree are very similar to those of a  $k^2$ -tree. Figure 2 illustrates how a quadtree’s recursive partitioning works.



**Figure 2.** A graph of 6 nodes (top) with its 8 × 8 binary adjacency matrix at various stages of recursive partitioning. At the bottom, a  $k^2$ -trees ( $k=2$ ) is constructed from the matrix.

### 2.2. LOUDS

$k^2$ -tree is based on unary encoding and LOUDS, which is a compact data structure introduced by Guy Jacobson in his paper and thesis [2,3]. A bit string is formed by a breadth-first traversal (going from left to right) of an ordinal (rooted, ordered) tree structure. Each parent node is encoded with a string of ‘1’ bits whose length indicates the number of children it has and each string ends with a ‘0’ bit. If the parent node has no children, only a single ‘0’ bit suffices.

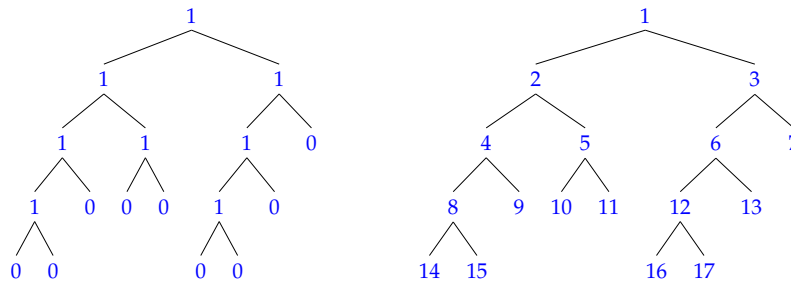
The parent and child relationship can be computed by two cornerstone functions for compact data structures: rank and select. These functions give us information about the node’s first-child, next-sibling(s), and parent, without the need of using pointers. They are described below:

- rank<sub>*b*</sub>(*m*) returns the number of bits which are set to *b*, left of position *m* (inclusive) in the bitmap where *b* is 0 or 1.
- select<sub>*b*</sub>(*i*) returns the position of the *i*-th *b* bit in the bitmap where *b* is 0 or 1.

By default, *b* is 1, i.e., rank(*m*) = rank<sub>1</sub>(*m*). These operations are inverses of each other. In other words, rank(select(*m*)) = select(rank(*m*)) = *m*. Since a linear scan is required to process the rank and select functions, the worst-case time complexity will be  $\mathcal{O}(n)$ .

To clarify how these functions work, consider the binary trees depicted in Figure 3 where the one on the left shows the values and the one on the right shows the numbering of the same tree. If the node has two children, it will be set to 1. Otherwise, it is set to 0. The values of this tree are put in a bit string shown in Figure 4. Figure 5 shows how the position of the left child, right child or parent of a certain node *m* is computed with the rank and select functions. An example follows:

To find the left child of node 8, we first need to compute rank(8), which is the total number of 1's from node 1 up to and including node 8 and the answer is 7. Therefore, the left child is located in  $2 \cdot \text{rank}(8) = 2 \cdot 7 = 14$  and the right child is in  $2 \cdot \text{rank}(8) + 1 = 2 \cdot 7 + 1 = 15$ . The parent of node 8 can be found by computing  $\text{select}(\lfloor 8/2 \rfloor)$  or  $\text{select}(\lfloor 4 \rfloor)$ . The answer can be arrived at by counting the total number of bits starting from node 1, skipping the ones with '0' bits. When we get to node 4 which gives us a total bit count of 4, we then know that node 4 is where the parent of node 8 is.



**Figure 3.** A binary tree example for LOUDs. The one on the left shows the values of the nodes and the one on the right shows the same tree with the numbering of the nodes in a left-to-right order. In this case the numbering starts with 1 at the root.

<i>m</i>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
bit	1	1	1	1	1	1	0	1	0	0	0	1	0	0	0	0	0

**Figure 4.** A bit string with the values from the binary tree in Figure 3.

<i>m</i>	1	2	3	4	5	6	8	12
Left child( $m = 2 \cdot \text{rank}(m)$ )	2	4	6	8	10	12	14	16
Right child( $m = 2 \cdot \text{rank}(m) + 1$ )	3	5	7	9	11	13	15	17
Parent( $m = \text{select}(\lfloor m/2 \rfloor)$ )	-	1	1	2	2	3	4	6

**Figure 5.** With the rank and select functions listed in the first column, we can navigate the binary tree in Figure 3 and compute the position node for the left child, right child or parent of the node.

In the next section, we will explain how the rank function can be used to determine the children's positions in a  $k^2$ -tree, thus enabling us to query the values of the cells.

### 2.3. $k^2$ -Tree

Originally proposed for compressing Web graphs,  $k^2$ -tree is a LOUDS variant compact data structure [22]. The tree represents a binary adjacency matrix of a graph (see Figure 2). It is constructed by recursively partitioning the matrix into square submatrices of equal size until each submatrix reaches a size of  $k \times k$  where  $k \geq 2$ . During the process of partitioning, if there is at least one cell in the submatrix that has a value of 1, the node in the tree will be set to 1. Otherwise, it will be set to 0 (i.e., it is a leaf and has no children) and this particular submatrix will not be partitioned any further. Figure 2 illustrates an example of a graph of 6 nodes, its  $8 \times 8$  binary adjacency matrix at various stages of recursive partitioning, and the  $k^2$ -tree that is constructed from the matrix.

The values of  $k^2$ -trees are basically stored in two bitmaps denoted by *T* (tree) and *L* (leaves). The values are traversed in a breadth-first fashion starting with the first level. The *T* bitmap stores the bits at all levels except the last one where its bits will be stored in the *L* bitmap. Note that the bit values of *T* which are either 0 or 1 will be stored as a bit vector. To illustrate this with an example, we again make use of the binary matrix in Figure 2. The *T* bitmap contains all the bits from levels 1 and 2. Thus the *T* bitmap has the following bits: 1001 1101 1000 (see Figure 6). The bits from the last level, level 3, will be stored in the *L* bitmap with the following bits: 1001 0001 1011 1111.

Consider a set *S* with elements from 1 to *n*, to find the child's or the parent's position of a certain node *m* in a  $k^2$ -tree, we perform the following operations:



$$\text{first-child}(m) \leftarrow \text{rank}(m) \cdot k^2 \text{ where } 1 \leq m \leq \|S\|$$

$$\text{parent}(m) \leftarrow \text{select}(\lfloor m/k^2 \rfloor) \text{ where } 1 \leq m \leq \|S\|$$

Once again using the  $k^2$ -tree in Figure 2 as an example, with the  $T$  bitmap (Figure 6) and the rank and select functions, we can navigate the tree and obtain the positions of the first child and the parent. Figure 7 shows how the nodes of the  $k^2$ -tree are numbered.

- Ex. Locate the first child of node 8:  
 $\text{rank}_1(8) * 4 = 6 * 4 = 24$   
 (There are 6 one bits in the  $T$  bitmap starting from node 0 up to and including node 8.)
- Ex. Locate the parent of node 11:  
 $\text{select}_1(\lfloor 11/4 \rfloor) = \text{select}_1(2) = 3$   
 (Start counting from node 0, skipping all nodes with '0' bits, and node 3 is the first node that gives a total number of 1-bit count of 2. Therefore, node 3 is the parent.)

Node	0	1	2	3	4	5	6	7	8	9	10	11
Bit	1	0	0	1	1	1	0	1	1	0	0	0

Figure 6. A  $T$  bitmap with the first node labeled as 0.

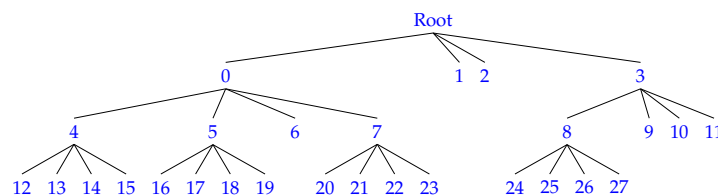


Figure 7. An example showing how the rank function is computed to obtain the children’s position on a  $k^2$ -tree node ( $k=2$ ) based on the tree in Figure 2. It starts with 0 on the first child of the root (first level) and the numbering traverses from left to right and from top to bottom.

It was shown that  $k^2$ -tree gave the best performance when the matrix was sparse with large clusters of 0’s or 1’s [20].

#### 2.4. DACs

This section describes DACs which is used in  $k^2$ -raster to directly access variable-length codes. Based on the concept of compact data structures, DACs were proposed in the papers published by Brisaboa et al. in 2009 and 2013 [23,24] and the structure was proven to yield good compression ratios for variable-length integer sequences. By means of the rank function, it gains fast direct access to any position of the sequence in a very compact space. The original authors also asserted that it was better suited for a sequence of integers with a skewed frequency distribution toward smaller integer values.

Different types of encoding are used for DACs and the one that we are interested in for  $k^2$ -raster is called Vbyte coding. Consider a sequence of integers  $x$ . Each integer, which is represented by  $\lfloor \log_2 x_i \rfloor + 1$  bits, is broken into blocks of bits of size  $S$ . Each block is stored as a chunk of  $S + 1$  bits. The chunk that holds the most significant bits has the highest bit set to 0 while the other chunks have their highest bit set to 1. For example, if we have an integer 20 ( $10100_2$ ) which is 5 bits long and if the block size is  $S = 3$ , then we can have 2 chunks denoted by the following: 0010 1100.

To show how the chunks are organized and stored, we again illustrate it with an example. If we have 3 integers of variable length 20 ( $10100_2$ ), 6 ( $110_2$ ), 73 ( $1001001_2$ ) and each block size is 3, then the three integers have the following representations.

- 20 0010 1100 ( $B_{1,2}A_{1,2} B_{1,1}A_{1,1}$ )
- 6 0110 ( $B_{2,1}A_{2,1}$ )
- 73 0001 1001 1001 ( $B_{3,3}A_{3,3} B_{3,2}A_{3,2} B_{3,1}A_{3,1}$ )

We will store them in three chunks of arrays  $A$  and bitmaps  $B$ . This is depicted in Figure 8. To retrieve the values in the arrays  $A$ , we make use of the corresponding bitmaps  $B$  with the rank function.

More information on DACs and the software code can be found in the papers [23,24].

$C_1$	$A_1$	100 ( $A_{1,1}$ )	110 ( $A_{2,1}$ )	001 ( $A_{3,1}$ )
	$B_1$	1 ( $B_{1,1}$ )	0 ( $B_{2,1}$ )	1 ( $B_{3,1}$ )
$C_2$	$A_2$	010 ( $A_{1,2}$ )	001 ( $A_{3,2}$ )	
	$B_2$	0 ( $B_{1,2}$ )	1 ( $B_{3,2}$ )	
$C_3$	$A_3$	001 ( $A_{3,3}$ )		
	$B_3$	0 ( $B_{3,3}$ )		

**Figure 8.** Organization of 3 Directly Addressable Codes (DACs) clusters.

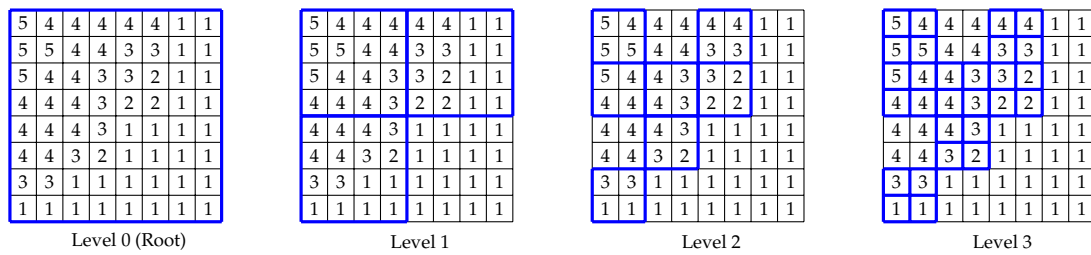
### 2.5. $k^2$ -Raster

$k^2$ -raster is a compact data structure that allows us to store raster pixels in reduced space. It consists of several basic components: bitmaps, DACs and LOUDS. Similar to a  $k^2$ -tree, the image matrix is partitioned recursively until each subquadrant is of size  $k^2$ . The resulting LOUDS tree topology contains the bitmap  $T$  where the elements are accessed with the rank function. Unlike  $k^2$ -tree, at each tree level, the maximum and minimum values of each subquadrant are stored in two bitmaps which are respectively called  $V_{max}$  and  $V_{min}$ . However, to compress the structure further, the maximum and minimum values of each level are compared with the corresponding values of the parent and their differences will replace the stored values in the  $V_{max}$  and  $V_{min}$  bitmaps. The rationale behind all this is to obtain smaller values for each node so as to get a better compression with DACs. An example of a simple  $8 \times 8$  matrix is given to illustrate this point in Figure 9. A  $k^2$ -raster is constructed from this matrix with maximum and minimum values stored in each node in Figure 10. The structure is further modified, according to the above discussion, to form a tree with smaller maximum and minimum values and this is shown in Figure 11.

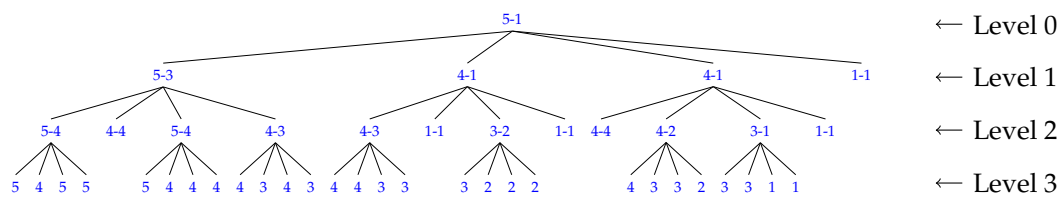
Next, with the exception of the root node at the top level, the  $V_{max}$  and  $V_{min}$  bitmaps at all levels are concatenated to form  $L_{max}$  and  $L_{min}$  bitmaps. The root's maximum ( $rMax$ ) and minimum ( $rMin$ ) values are integer values and will remain uncompressed.

For an image of size  $n \times n$  with  $n$  bands, the time complexity to build all the  $k^2$ -rasters is  $\mathcal{O}(n^3)$  [22]. To query a cell from the structure, which has a tree height of at most  $\lceil \log_k n \rceil$  levels, the time complexity to extract a codeword at a single  $L_{max}$  level is  $\mathcal{O}(\log_k n)$ , and this is the worst-case time to traverse from the root node to the last level of the structure. The number of levels,  $\mathcal{L}$ , in  $L_{max}$  can be obtained from the maximum integer in the sequence and with this, we can compute the time complexity for a cell query, which is  $\mathcal{O}(\log_k n \cdot \mathcal{L})$  [23,25].

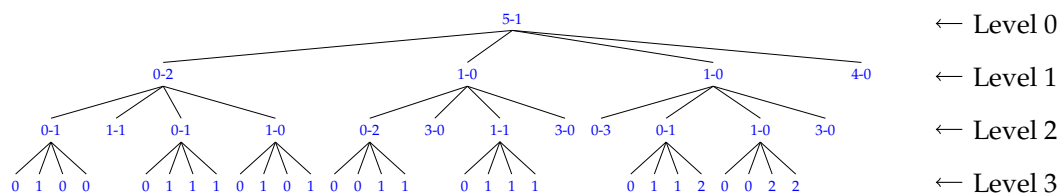
To sum up, a  $k^2$ -raster structure is composed of a bitmap  $T$ , a maximum bitmap  $L_{max}$ , a minimum bitmap  $L_{min}$ , a root maximum  $rMax$  integer value and a root minimum  $rMin$  integer value.



**Figure 9.** An example of an  $8 \times 8$  matrix for  $k^2$ -raster. The matrix is recursively partitioned into square subquadrants of equal size. During the process, unless all the cells in a subquadrant have the same value, the partitioning will continue. Otherwise the partitioning of this particular subquadrant will end at this point.



**Figure 10.** A  $k^2$ -raster ( $k = 2$ ) tree storing the maximum and minimum values for each quadrant of every recursive subdivision of the matrix in Figure 9. Every node contains the maximum and minimum values of the subquadrant, separated by a dash. On the last level, only one value is shown as each subquadrant contains only one cell.



**Figure 11.** Based on the tree in Figure 10, the maximum value of each node is subtracted from that of its parent while the minimum value of the parent is subtracted from the node’s minimum value. These differences will replace their corresponding values in the node. The maximum and minimum values of the root remain the same.

2.6. Predictive Method

As mentioned in the Introduction, an interband predictor called 3D-CALIC was proposed by Wu et al. in 2000 and another predictor called M-CALIC by Magli et al. in 2004. Our predictor is based on the idea of least squares method and the use of reference bands that were discussed in both the 3D-CALIC [14] and M-CALIC [15] papers. Consider two neighboring or close neighboring bands of the same hyperspectral image. These bands can be represented by two vectors  $\mathbf{X} = (x_1, x_2, x_3, \dots, x_{n-1}, x_n)$  and  $\mathbf{Y} = (y_1, y_2, y_3, \dots, y_{n-1}, y_n)$  where  $x_i$  and  $y_i$  are two pixels that are located at the same spatial position but in different bands, and  $n$  is the number of pixels in each band. We can then employ the close similarity between the bands to predict the pixel value in the current band  $\mathbf{Y}$  using the corresponding pixel value in band  $\mathbf{X}$ , which we designate as the reference band.

A predictor for a particular band can be built from the linear equation:

$$\hat{\mathbf{Y}} = \alpha\mathbf{X} + \beta \tag{1}$$

so as to minimize  $\|\hat{\mathbf{Y}} - \mathbf{Y}\|_2^2$  where  $\hat{\mathbf{Y}}$  is the predicted value and  $\mathbf{Y}$  is the actual value of the current band. The optimal values for  $\alpha$  and  $\beta$  should minimize the prediction error of the current pixel and can be obtained by using the least squares solution:

$$\hat{\alpha} = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{n \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2}, \quad (2)$$

$$\hat{\beta} = \frac{n \sum_{i=1}^n y_i - \hat{\alpha} \sum_{i=1}^n x_i}{n} \quad (3)$$

where  $n$  is the size of each band, i.e., the height multiplied by the width,  $\hat{\alpha}$  the optimal value of  $\alpha$  and  $\hat{\beta}$  the optimal value of  $\beta$ .

The difference between the actual and predicted pixel values of a band is known as the residual value or the prediction error. When all the pixel values in the current band are calculated, these prediction residuals will be saved in a vector, which will later be used as input to a  $k^2$ -raster.

In other words, for a particular pixel in the current band and the corresponding pixel in the reference band,  $\delta_i$  being the residual value,  $y_i$  the actual value of the current band, and  $x_i$  the value of the reference band, to encode, the following equation is used:

$$\delta_i = y_i - (\hat{\alpha} \cdot x_i + \hat{\beta}). \quad (4)$$

To decode, the following equation is used:

$$y_i = \delta_i + (\hat{\alpha} \cdot x_i + \hat{\beta}). \quad (5)$$

The distance from the reference band affects the residual values. The closer the current band is to the reference band, the smaller the residual values would tend to be. We can arrange the bands into groups. For example, the first band can be chosen as the reference and the second, third and fourth bands will have their residual values calculated with respect to the first band. And the next group starts with the fifth band as the reference band, etc.

For this coding method, the group size (stored as a 2-byte short integer) as well as the  $\hat{\alpha}$  and  $\hat{\beta}$  values for each band (stored as 8-byte double's) will need to be saved for use in both the encoder and the decoder. Note that the size of these extra data is insignificant - which generally comes to around 3.5 kB - compared to the overall size of the structure.

### 2.7. Differential Method

In the differential encoding, which is a special case of the predictor where  $\alpha = 1$  and  $\beta = 0$ , the residual value is obtained by simply taking the difference between the reference band and the current band. For a particular pixel in the current band and the corresponding pixel in the reference band,  $\delta_i$  being the residual value,  $y_i$  the actual value of the current band, and  $x_i$  the value of the reference band, to encode, the following equation is used:

$$\delta_i = y_i - x_i. \quad (6)$$

To decode, the following equation is used:

$$y_i = \delta_i + x_i. \quad (7)$$

Like the predictor, we can use the first band as the reference band and the next several bands can use this reference band to find the residual values. Again, the grouping is repeated up to the last band. For this coding method, only the group size (stored as a 2-byte short integer) needs to be saved.

## 2.8. Related Work

Since the publication of the proposals on  $k^2$ -tree and  $k^2$ -raster, more research has been done to extend the capabilities of the structures to 3D where the first and second dimensions represent the spatial element and the third dimension the time element.

Based on their previous research of  $k^2$ -raster, Silva-Coira et al. [26] proposed a structure called Temporal  $k^2$ -raster (T- $k^2$ -raster) which represents a time-series of rasters. It takes advantage of the fact that in a time-series, the values in a matrix  $M_1$  are very close to, if not the same as, the next matrix  $M_2$  or even the one after that,  $M_3$ , along the timeline. The matrices can then be grouped into  $\tau$  time instants where the values of the elements of the first matrix in the group is subtracted from the corresponding ones in the current matrix. The result will be smaller integer values that would help form a more compact tree as there are likely to be more zeros in the tree than before. Their experimental results bear this out. When the  $\tau$  value is small ( $\tau = 4$ ), the sizes are small. However, as would be expected, the results are not as favorable when the  $\tau$  value becomes larger ( $\tau = 50$ ). Akin to the Temporal  $k^2$ -raster, the differential encoding on  $k^2$ -raster that we are proposing in this paper also exploits the similarity between neighboring matrices or bands in a hyperspectral image to form a more compact structure.

Another study on compact representation of raster images in a time-series was proposed earlier this year by Cruces et al. in [27]. This method is based on the 3D to 2D mapping of a raster where 3D tuples  $\langle x, y, z \rangle$  are mapped into a 2D binary grid. That is, a raster of size  $w \times h$  with values in a certain range, between 0 and  $v$  inclusive will have a binary matrix of  $w \times h$  columns and  $v+1$  rows. All the rasters will then be concatenated into a 3D cube and stored as a  $k^3$ -tree.

## 3. Results

In this section we describe some of the experiments that were performed to show the use of compact data structures, prediction and differential encoding for real-time processing and compression. First, we show the results with other compression algorithms and techniques that are currently in use such as gzip, bzip2, xz, M-CALIC [15] and CCSDS 123.0-B-1 [9]. Then we compare the build time and the data access time for  $k^2$ -raster with and without prediction and differential encoding. Next, we show the results of different rates in  $k^2$ -raster that are produced as different  $k$ -values are applied. Similarly, the results of different group sizes for prediction and differential encoding are shown. Finally, the predictive method and the differential method are compared.

Experiments were conducted using hyperspectral images from different sensors: Atmospheric Infrared Sounder (AIRS), Airborne Visible/Infrared Imaging Spectrometer (AVIRIS), Compact Reconnaissance Imaging Spectrometer for Mars (CRISM), Hyperion, and Infrared Atmospheric Sounding Interferometer (IASI). Except for IASI, all of them are publicly available for download (<http://cwe.ccsds.org/sls/docs/sls-dc/123.0-B-Info/TestData>). Table 1 gives more detailed information on these images. The table also shows the bit-rate reduction for using  $k^2$ -raster with and without prediction. Performance in terms of bit rate and entropy is evaluated for them.

For best results in  $k^2$ -raster for the testing images, we used the optimal  $k$ -value, and also in the case of the predictor and the differential encoder, the optimal group size for each image was used. The effects of using different  $k$ -values and different group sizes will be discussed and tested in two of the subsections below.

To build the structure of  $k^2$ -raster and the cell query functions, a program in C was written. The algorithms presented in the paper by Ladra et al. [20] were the basis and reference for writing the code. The DACs software that was used in conjunction with our program is available at the Universidade da Coruña's Database Laboratory (Laboratorio de Bases de Datos) website (<http://lbd.udc.es/research/DACS/>). The package is called "DACs, optimization with no further restrictions". As for the predictive and differential methods, another C program was written to perform the tasks needed to give us the results that we will discuss below. All the code was compiled using gcc or g++ 5.4.0 20160609 with -Ofast optimization.

**Table 1.** Hyperspectral images used in our experiments. It also shows the bit rate and bit rate reduction using  $k^2$ -raster with and without the predictor.  $x$  is the image width,  $y$  the image height and  $z$  the number of spectral bands. The unit bpppb stands for bits per pixel per band.

Sensor	Name	C/U*	Acronym	Original Dimensions ( $x \times y \times z$ )	Bit Depth (bpppb)	Optimal $k$ -Value	$k^2$ -Raster Bit Rate (bpppb)	$k^2$ -Raster Bit-Rate Reduction (%)	$k^2$ -Raster+ Predictor Bit Rate (bpppb)	$k^2$ -Raster+ Predictor Bit-Rate Reduction (%)
AIRS	9	U	AG9	$90 \times 135 \times 1501$	12	6	9.49	21%	6.76	44%
	16	U	AG16	$90 \times 135 \times 1501$	12	6	9.12	24%	6.63	45%
	60	U	AG60	$90 \times 135 \times 1501$	12	6	9.81	18%	7.06	41%
	126	U	AG126	$90 \times 135 \times 1501$	12	6	9.61	20%	7.05	41%
	129	U	AG129	$90 \times 135 \times 1501$	12	6	8.65	28%	6.47	46%
	151	U	AG151	$90 \times 135 \times 1501$	12	6	9.53	21%	7.02	41%
	182	U	AG182	$90 \times 135 \times 1501$	12	6	9.68	19%	7.19	40%
	193	U	AG193	$90 \times 135 \times 1501$	12	6	9.44	21%	7.06	41%
AVIRIS	Yellowstone sc. 00	C	ACY00	$677 \times 512 \times 224$	16	6	9.61	40%	6.87	57%
	Yellowstone sc. 03	C	ACY03	$677 \times 512 \times 224$	16	6	9.42	41%	6.72	58%
	Yellowstone sc. 10	C	ACY10	$677 \times 512 \times 224$	16	4	7.57	53%	5.84	64%
	Yellowstone sc. 11	C	ACY11	$677 \times 512 \times 224$	16	6	8.81	45%	6.52	59%
	Yellowstone sc. 18	C	ACY18	$677 \times 512 \times 224$	16	6	9.78	39%	7.04	56%
	Yellowstone sc. 00	U	AUY00	$680 \times 512 \times 224$	16	9	11.92	25%	9.04	44%
	Yellowstone sc. 03	U	AUY03	$680 \times 512 \times 224$	16	9	11.74	27%	8.87	45%
	Yellowstone sc. 10	U	AUY10	$680 \times 512 \times 224$	16	9	9.99	38%	8.00	50%
	Yellowstone sc. 11	U	AUY11	$680 \times 512 \times 224$	16	9	11.27	30%	8.77	45%
	Yellowstone sc. 18	U	AUY18	$680 \times 512 \times 224$	16	9	12.15	24%	9.29	42%
CRISM	frt000065e6_07_sc164	U	C164	$640 \times 420 \times 545$	12	6	10.08	16%	10.02	16%
	frt00008849_07_sc165	U	C165	$640 \times 450 \times 545$	12	6	10.37	14%	10.33	14%
	frt0001077d_07_sc166	U	C166	$640 \times 480 \times 545$	12	6	11.05	8%	11.08	8%
	hr100004f38_07_sc181	U	C181	$320 \times 420 \times 545$	12	5	9.97	17%	9.52	21%
	hr10000648f_07_sc182	U	C182	$320 \times 450 \times 545$	12	5	10.11	16%	9.84	18%
	hr10000ba9c_07_sc183	U	C183	$320 \times 480 \times 545$	12	5	10.65	11%	10.59	12%

Table 1. Cont.

Sensor	Name	C/U*	Acronym	Original Dimensions ( $x \times y \times z$ )	Bit Depth (bpppb)	Optimal $k$ -Value	$k^2$ -raster Bit Rate (bpppb)	$k^2$ -Raster Bit-Rate Reduction (%)	$k^2$ -Raster+ Predictor Bit Rate (bpppb)	$k^2$ -Raster+ Predictor Bit-Rate Reduction (%)
Hyperion	Agricultural 2905 †	C	HCA1	256 × 2905 × 242	12	8	8.20	32%	7.47	38%
	Agricultural 3129 †	C	HCA2	256 × 3129 × 242	12	8	8.08	33%	7.50	37%
	Coral Reef †	C	HCC	256 × 3127 × 242	12	8	7.38	39%	7.41	38%
	Urban †	C	HCU	256 × 2905 × 242	12	8	8.59	28%	7.83	35%
	Filtered Erta Ale †	U	HFUEA	256 × 3187 × 242	12	8	6.84	43%	5.99	50%
	Filtered Lake Monona †	U	HFULM	256 × 3176 × 242	12	8	6.79	43%	6.06	49%
	Filtered Mt. St. Helena †	U	HFUMS	256 × 3242 × 242	12	8	6.78	43%	5.88	51%
	Erta Ale †	U	HUEA	256 × 3187 × 242	12	8	7.57	37%	6.99	42%
	Lake Monona †	U	HULM	256 × 3176 × 242	12	8	7.52	37%	7.08	41%
	Mt. St. Helena †	U	HUMS	256 × 3242 × 242	12	8	7.49	38%	6.93	42%
IASI	Level 0 1 ‡	U	I01	60 × 1528 × 8359	12	4	5.93	51%	4.69	61%
	Level 0 2 ‡	U	I02	60 × 1528 × 8359	12	4	5.90	51%	4.75	60%
	Level 0 3 ‡	U	I03	60 × 1528 × 8359	12	4	5.42	55%	4.58	62%
	Level 0 4 ‡	U	I04	60 × 1528 × 8359	12	4	6.23	48%	4.90	59%

\*: Calibrated or Uncalibrated; †: Cropped to 256 × 512 × 242; ‡: Cropped to 60 × 256 × 8359.

The machine that these experiments ran on has an Intel Core 2 Duo CPU E7400 @2.80GHz with 3072KB of cache and 3GB of RAM. The operating system is Ubuntu 16.04.5 LTS with kernel 4.15.0-47-generic (64 bits).

To ensure that there was no loss of information, the image was reconstructed by reverse transformation and verified to be identical to the original image in the case of predictive and differential methods. For  $k^2$ -raster, after saving the structure to disk, we made sure that the original image could be reconstructed from the saved data.

### 3.1. Comparison with Other Compression Algorithms

Both  $k^2$ -raster with and without predictive and differential encoding were compared to other commonly-used compression algorithms such as gzip, bzip2, xz, and specialized algorithms such as M-CALIC and CCSDS 123.0-B-1. The results for the comparison are shown in Table 2 and depicted in Figure 12.

It can be seen that  $k^2$ -raster alone already performed better than gzip. When it was used with the predictor, it produced a bit rate that was basically on a par with and sometimes better than other compression algorithms such as xz or bzip2. However, it could not attain the bit-rate level done by CCSDS 123.0-B-1 or M-CALIC. This was to be expected as both are specialized compression techniques, and CCSDS 123.0-B-1 is considered a baseline against which all compression algorithms for hyperspectral images are measured. Nevertheless,  $k^2$ -raster provides direct access to the elements without full decompression, and this is undoubtedly the major advantage it has over all the aforementioned compression algorithms.

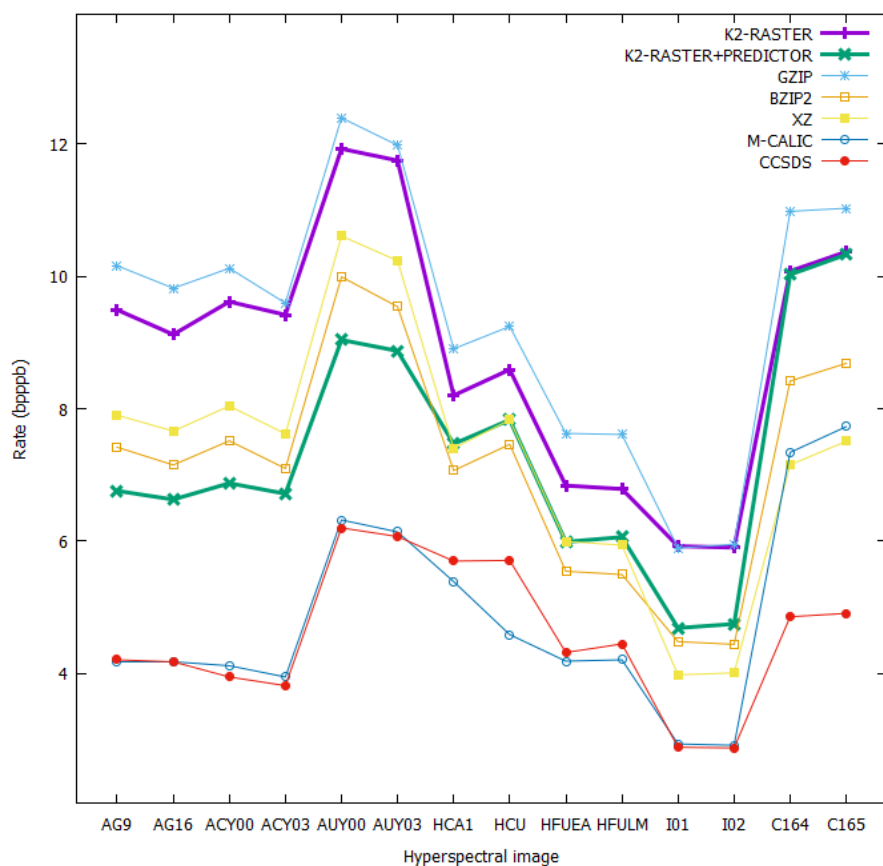


Figure 12. A rate (bpppb) comparison with other compression techniques.



**Table 2.** A rate (bppb) comparison with other compression techniques. The optimal values for all compression algorithms (except for M-CALIC, CCSDS 123.0-B-1) are highlighted in red. Results for CCSDS 123.0-B-1 are from [28].

Sensor	Name	C/U *	Acronym	Compression Technique (bppb)							
				$k^2$ -Raster	$k^2$ -Raster + Predictor	$k^2$ -Raster + Differential	gzip	bzip2	xz	M-CALIC	CCSDS 123.0-B-1
AIRS	9	U	AG9	9.49	6.76	7.52	10.16	7.42	7.90	4.19	4.21
	16	U	AG16	9.12	6.63	7.29	9.82	7.15	7.66	4.19	4.18
	60	U	AG60	9.81	7.06	7.82	10.53	7.71	8.23	4.41	4.36
	126	U	AG126	9.61	7.05	7.78	10.33	7.64	8.10	4.39	4.38
	129	U	AG129	8.65	6.47	6.96	9.50	6.68	7.22	4.08	4.12
	151	U	AG151	9.53	7.02	7.74	10.31	7.43	7.97	4.39	4.41
	182	U	AG182	9.68	7.19	7.94	10.64	7.79	8.33	4.45	4.42
	193	U	AG193	9.44	7.06	7.77	10.15	7.47	7.94	4.42	4.42
AVIRIS	Yellowstone sc. 00	C	ACY00	9.61	6.87	7.79	10.12	7.51	8.04	4.12	3.95
	Yellowstone sc. 03	C	ACY03	9.42	6.72	7.65	9.59	7.10	7.62	3.95	3.82
	Yellowstone sc. 10	C	ACY10	7.57	5.84	6.26	7.41	5.30	5.73	3.31	3.36
	Yellowstone sc. 11	C	ACY11	8.81	6.52	6.85	9.04	6.65	7.07	3.71	3.63
	Yellowstone sc. 18	C	ACY18	9.78	7.04	7.53	10.00	7.45	7.95	4.09	3.90
	Yellowstone sc. 00	U	AUY00	11.92	9.04	10.04	12.39	9.99	10.61	6.32	6.20
	Yellowstone sc. 03	U	AUY03	11.74	8.87	9.91	11.98	9.54	10.23	6.14	6.07
	Yellowstone sc. 10	U	AUY10	9.99	8.00	8.57	10.17	7.71	8.40	5.53	5.58
	Yellowstone sc. 11	U	AUY11	11.27	8.77	9.21	11.49	9.08	9.66	5.91	5.84
	Yellowstone sc. 18	U	AUY18	12.15	9.29	9.92	12.29	9.90	10.58	6.33	6.21
CRISM	frt000065e6_07_sc164	U	C164	10.08	10.02	10.06	10.98	8.42	7.15	7.34	4.86
	frt00008849_07_sc165	U	C165	10.37	10.33	10.37	11.03	8.68	7.51	7.73	4.91
	frt0001077d_07_sc166	U	C166	11.05	11.08	11.14	11.20	9.04	7.64	8.44	5.44
	hrl00004f38_07_sc181	U	C181	9.97	9.52	9.52	10.77	8.28	8.20	7.09	4.27
	hrl0000648f_07_sc182	U	C182	10.11	9.84	9.86	10.90	8.53	7.90	7.28	4.49
	hrl0000ba9c_07_sc183	U	C183	10.65	10.59	10.64	10.87	8.52	7.28	7.91	4.96

Table 2. Cont.

Sensor	Name	C/U *	Acronym	Compression Technique (bpppb)							CCSDS 123.0-B-1
				$k^2$ -Raster	$k^2$ -Raster + Predictor	$k^2$ -Raster + Differential	gzip	bzip2	xz	M-CALIC	
Hyperion	Agricultural 2905 †	C	HCA1	8.20	7.47	7.47	8.90	7.07	7.40	5.39	-
	Agricultural 3129 †	C	HCA2	8.08	7.50	7.50	8.84	7.04	7.35	5.28	5.70
	Coral Reef †	C	HCC	7.38	7.41	7.41	7.45	5.74	5.90	4.59	5.42
	Urban †	C	HCU	8.59	7.83	7.83	9.24	7.46	7.83	5.25	5.71
	Filtered Erta Ale †	U	HFUEA	6.84	5.99	6.15	7.63	5.55	6.00	4.19	4.32
	Filtered Lake Monona †	U	HFULM	6.79	6.06	6.18	7.61	5.50	5.94	4.21	4.45
	Filtered Mt. St. Helena †	U	HFUMS	6.78	5.88	6.15	7.18	5.44	5.74	4.11	4.35
	Erta Ale †	U	HUEA	7.57	6.99	7.06	8.69	6.41	6.73	4.87	4.32
	Lake Monona †	U	HULM	7.52	7.08	7.13	8.69	6.46	6.74	4.94	4.45
Mt. St. Helena †	U	HUMS	7.49	6.93	7.04	8.26	6.28	6.48	4.82	4.36	
IASI	Level 0 1 ‡	U	I01	5.93	4.69	5.01	5.90	4.48	3.98	2.94	2.89
	Level 0 2 ‡	U	I02	5.90	4.75	5.03	5.96	4.44	4.01	2.92	2.88
	Level 0 3 ‡	U	I03	5.42	4.58	4.79	5.25	3.94	3.75	2.92	2.88
	Level 0 4 ‡	U	I04	6.23	4.90	5.20	6.30	4.71	4.24	2.97	2.90

\*: Calibrated or Uncalibrated; †: Cropped to  $256 \times 512 \times 242$  except for CCSDS 123.0; ‡: Cropped to  $60 \times 256 \times 8359$  except for CCSDS 123.0.

### 3.2. Build Time

Both the time to build the  $k^2$ -raster only and the time to build  $k^2$ -raster with predictive and differential preprocessing were measured. They were then compared against the time to compress the data with gzip. The results are presented in Table 3. We can see that the build time for  $k^2$ -raster only took half as long as with gzip. Comparing the predictive and the differential methods, the time difference is small although it generally took longer to build the former than the latter due to the additional time needed to compute the values of  $\hat{\alpha}$  and  $\hat{\beta}$ . Both, however, still took less time to build than gzip compression.

**Table 3.** A comparison of build time (in seconds) using  $k^2$ -raster only and  $k^2$ -raster with predictive and differential methods.

Hyperspectral Image	Build Time (s)			Gzip Compression (s)
	$k^2$ -Raster	$k^2$ -Raster + Predictor	$k^2$ -Raster + Differential	
AG9	1.86	2.23	2.12	3.18
AG16	1.78	2.22	2.09	3.49
ACY00	8.32	10.11	9.49	15.01
ACY03	8.26	10.00	9.47	15.32
AUY00	5.56	7.39	6.84	12.10
AUY03	5.59	7.38	6.76	12.68
C164	17.84	21.32	21.59	27.94
C165	17.89	22.83	22.92	30.83
HCA1	1.98	2.67	2.47	5.59
HCA2	1.98	2.64	2.42	5.80
HFUEA	2.38	3.01	3.05	7.59
HFULM	2.41	3.04	2.87	7.57
HFUMS	2.33	2.95	2.76	8.26
I01	14.58	18.62	16.56	31.59
I02	14.66	17.49	16.66	29.64

### 3.3. Access Time

Several tests were conducted to see what the access time was like to query the cells in each image and we found that the time for a random cell access took longer for a predictor compared to just using the  $k^2$ -raster. This was expected but we should bear in mind that the bit rates are reduced when a predictor is used, thus decreasing storage size and transmission rate. Note that the last column also lists the time to decompress a gzip image file and it took at least 4 or 5 times longer than using a predictor to randomly access the data  $10^5$  times. Table 4 shows the results of access time in milliseconds for 100,000 iterations of random cell query done by `getCell()`, a function which was described in the paper from Ladra et al. [20] for accessing pixel values in a  $k^2$ -raster.

**Table 4.** A comparison of access time (in milliseconds) using  $k^2$ -raster only and  $k^2$ -raster with predictive and differential encoders.

Hyperspectral Image	100,000 Iterations of Random Access (ms)			Gzip Decompression (ms)
	$k^2$ -Raster	$k^2$ -Raster + Predictor	$k^2$ -Raster + Differential	
AG9	90	125	92	474
AG16	85	121	85	459
ACY00	275	485	426	1949
ACY03	269	474	424	1912
AUY00	151	489	402	1941
AUY03	151	485	402	1957
C164	273	400	381	4048
C165	301	420	397	4382
HCA1	77	131	127	735
HCA2	76	121	118	737
HFUEA	93	150	129	684
HFULM	92	148	129	680
HFUMS	91	146	134	670
I01	155	222	244	2517
I02	168	236	255	2396

### 3.4. Use of Different $k$ -Values

With  $k^2$ -raster, we found that different  $k$ -values used in the structure would produce different bit rates and different access time. In general, for most of our testing images the  $k$ -value is at its optimal bit-rate level when it is between 4 and 9. The reason is that as the  $k$ -value increases, the height of the constructed tree becomes smaller. Therefore, the number of nodes in the tree will decrease and so will the size of the bitmaps  $L_{max}$  and  $L_{min}$  that need to be stored in the structure. Table 5 shows the bit rates of some of the testing images between  $k = 2$  and  $k = 20$ . Additionally, experiments show that as the  $k$ -value becomes higher, the access time also becomes shorter, as can be seen in Table 6. As the  $k$ -value gets larger, the tree becomes shorter, thus making it faster to traverse from the top level to a lower level when searching for a particular node in the tree. As there is a trade-off between storage size and access time, for the experiments, the  $k$ -value that produces the lowest bit rate for the image was used.

For those who would like to know which  $k$ -value would give the best or close to the best rate, we recommend them to use a value of 6 as a general rule. This can be seen from Table 5 where the difference in the rate produced by this value and the one by the optimal  $k$ -value averages out to be only about 0.19 bpppb.

### 3.5. Use of Different Group Sizes

Tests were performed to see how the group size affects the predictive and differential methods. The group sizes were 2, 4, 8, 12, 16, 20, 24, 28 and 32. The results in Table 7 and Figure 13 show that for most images, they are at their optimal bit rates when the size is 4 or 8. The best bit-rate values are highlighted in red. For the range of group size tested, we can also see that except for the CRISM scenes (which consist of pixels with low spatial correlation, thus leading to inaccurate prediction), the bit rates for the predictor are always lower than the ones for differential encoding, irrespective of the group size.

For users who are interested in knowing which group size is the best to apply to the predictive and differential methods, a size of 4 is recommended for general use as the difference in bit rate produced by this group size and the one by the optimal group size averages out to be about 0.06 bpppb.

For the rest of the experiments, the optimal group size for each image was used to obtain the bit rate.

**Table 5.** Rates (bpppb) for different  $k$ -values for some of the testing images. The  $k$ -value with the lowest rate is in red.

Hyperspectral Image	$k = 2$	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AG9	13.06	10.11	10.03	10.47	9.49	9.98	10.68	9.89	10.65	-	11.23	10.33	11.29	9.53	11.57	11.72	10.78	12.52	12.13
AG16	12.72	9.78	9.66	10.11	9.12	9.57	10.32	9.51	10.29	-	10.82	9.98	10.86	9.17	11.11	11.28	10.32	12.07	11.68
ACY00	12.34	10.20	9.76	-	9.61	9.91	-	9.69	9.83	9.87	9.95	10.24	10.20	-	-	-	-	-	-
ACY03	11.81	9.87	9.56	-	9.42	9.71	-	9.50	9.65	9.70	9.76	10.01	9.98	-	-	-	-	-	-
AUY00	15.31	12.93	12.20	-	12.08	12.35	-	11.92	12.11	12.13	12.17	12.52	12.43	-	-	-	-	-	-
AUY03	15.03	12.60	12.00	-	11.90	12.20	-	11.74	11.93	11.94	12.00	12.34	12.25	-	-	-	-	-	-
C164	12.60	10.42	10.17	-	10.08	-	-	10.34	10.20	10.76	10.48	-	-	-	-	-	-	-	-
C165	12.84	10.67	10.48	-	10.37	-	-	10.54	10.51	10.79	11.03	-	-	-	-	-	-	-	-
HCA1	10.79	9.41	8.85	8.45	8.74	9.36	8.20	8.51	8.68	8.85	8.88	8.92	9.21	-	-	-	-	-	-
HCC	9.43	8.12	7.79	7.41	7.75	8.40	7.38	7.67	7.85	8.06	8.12	8.26	8.56	-	-	-	-	-	-
HFUEA	8.82	7.80	7.30	7.24	7.41	8.07	6.84	7.25	7.43	7.66	7.68	7.71	8.07	-	-	-	-	-	-
HFULM	8.69	7.70	7.20	7.13	7.33	8.02	6.79	7.21	7.40	7.64	7.66	7.68	8.05	-	-	-	-	-	-
I01	8.03	-	5.93	-	-	6.45	-	-	-	-	-	-	-	-	6.59	7.79	8.30	8.73	6.36
I02	8.02	-	5.90	-	-	6.48	-	-	-	-	-	-	-	-	6.64	7.92	8.46	8.97	6.45

**Table 6.** Access time (ms) for different  $k$ -values for some of the testing images. The best access time is in red.

Hyperspectral Image	Access Time (ms)																		
	$k = 2$	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AG9	345	167	130	114	91	84	83	82	82	-	60	59	56	56	55	52	60	51	47
AG16	334	152	114	108	85	79	78	80	75	-	55	54	53	53	59	51	58	48	45
ACY00	3553	1085	573	-	291	225	-	152	133	125	114	110	104	-	-	-	-	-	-
ACY03	3521	1112	572	-	277	223	-	149	131	122	112	120	102	-	-	-	-	-	-
AUY00	3569	1135	592	-	292	228	-	153	133	126	115	113	106	-	-	-	-	-	-
AUY03	3559	1123	585	-	279	221	-	152	133	124	115	109	103	-	-	-	-	-	-
C164	2924	964	606	-	272	-	-	159	161	138	131	-	-	-	-	-	-	-	-
C165	3754	1017	555	-	290	-	-	178	156	152	145	-	-	-	-	-	-	-	-
HCA1	1179	384	213	154	124	106	80	78	69	71	62	61	62	-	-	-	-	-	-
HCC	1203	406	233	172	139	123	95	94	86	85	83	79	79	-	-	-	-	-	-
HFUEA	1409	465	262	184	148	127	93	95	89	84	77	76	76	-	-	-	-	-	-
HFULM	1427	467	262	193	155	130	94	96	87	90	79	81	80	-	-	-	-	-	-
I01	999	-	779	-	-	679	-	-	-	-	-	-	-	-	610	709	715	728	450
I02	1047	-	759	-	-	698	-	-	-	-	-	-	-	-	651	746	746	730	472

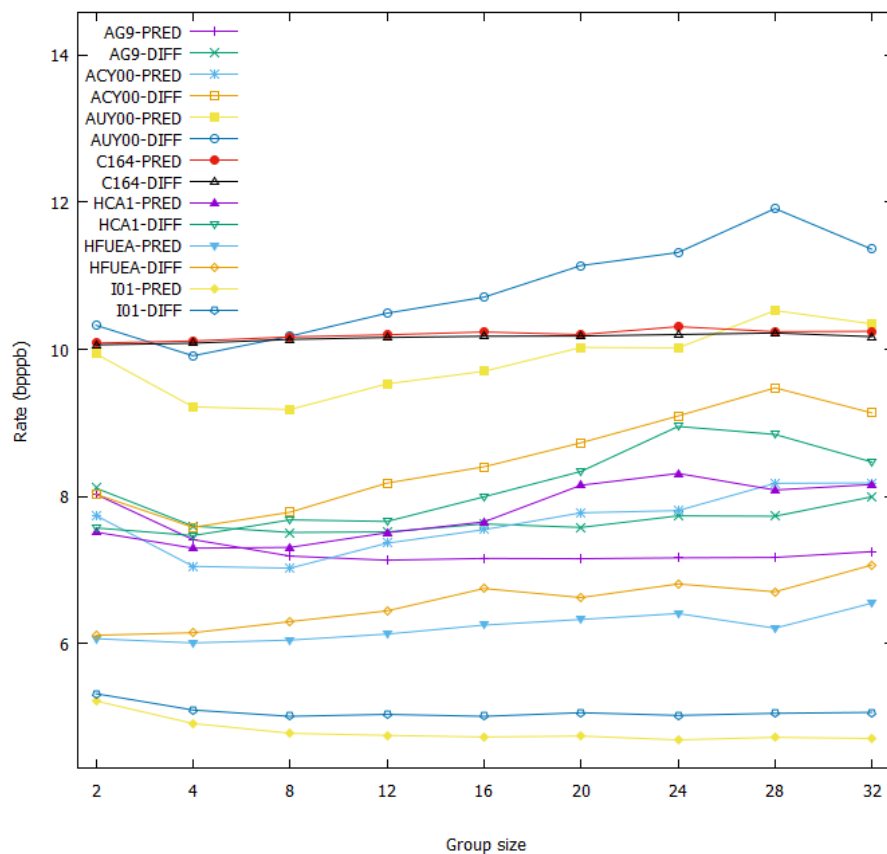


Figure 13. A rate (bpppb) comparison of different group sizes.

Table 7. A rate (bpppb) comparison of different group sizes using the predictive and the differential methods. The optimal values are highlighted in red.

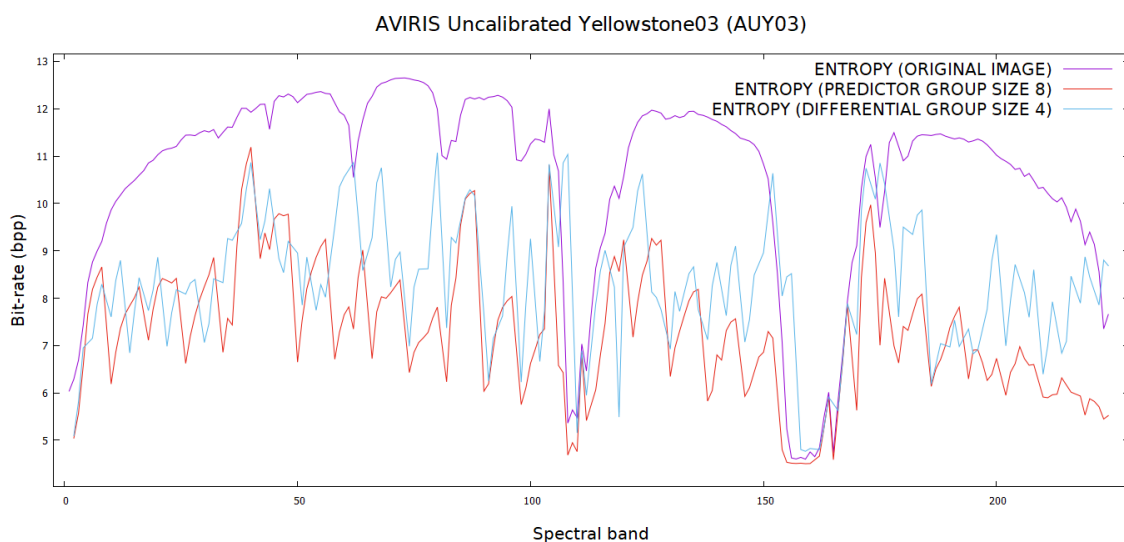
Hyperspectral Image	Group Size									
	2	4	8	12	16	20	24	28	32	
AG9-Pred	8.03	7.41	7.19	7.13	7.16	7.15	7.17	7.17	7.25	
AG9-Diff	8.11	7.60	7.51	7.52	7.63	7.58	7.74	7.73	8.00	
ACY00-Pred	7.74	7.05	7.03	7.36	7.55	7.78	7.81	8.18	8.18	
ACY00-Diff	8.03	7.58	7.79	8.18	8.40	8.73	9.09	9.48	9.13	
AUY00-Pred	9.94	9.22	9.18	9.53	9.70	10.03	10.02	10.53	10.34	
AUY00-Diff	10.33	9.91	10.18	10.49	10.71	11.14	11.32	11.91	11.36	
C164-Pred	10.08	10.11	10.17	10.20	10.23	10.20	10.31	10.24	10.24	
C164-Diff	10.06	10.08	10.13	10.16	10.18	10.20	10.31	10.22	10.17	
HCA1-Pred	7.51	7.30	7.31	7.51	7.65	8.15	8.31	8.09	8.16	
HCA1-Diff	7.57	7.47	7.68	7.66	8.00	8.34	8.95	8.84	8.47	
HFUEA-Pred	6.07	6.01	6.05	6.13	6.25	6.33	6.41	6.21	6.55	
HFUEA-Diff	6.11	6.15	6.30	6.44	6.75	6.63	6.81	6.70	7.07	
I01-Pred	5.22	4.91	4.78	4.75	4.73	4.74	4.69	4.73	4.71	
I01-Diff	5.32	5.10	5.01	5.04	5.01	5.06	5.02	5.05	5.06	

### 3.6. Predictive and Differential Methods

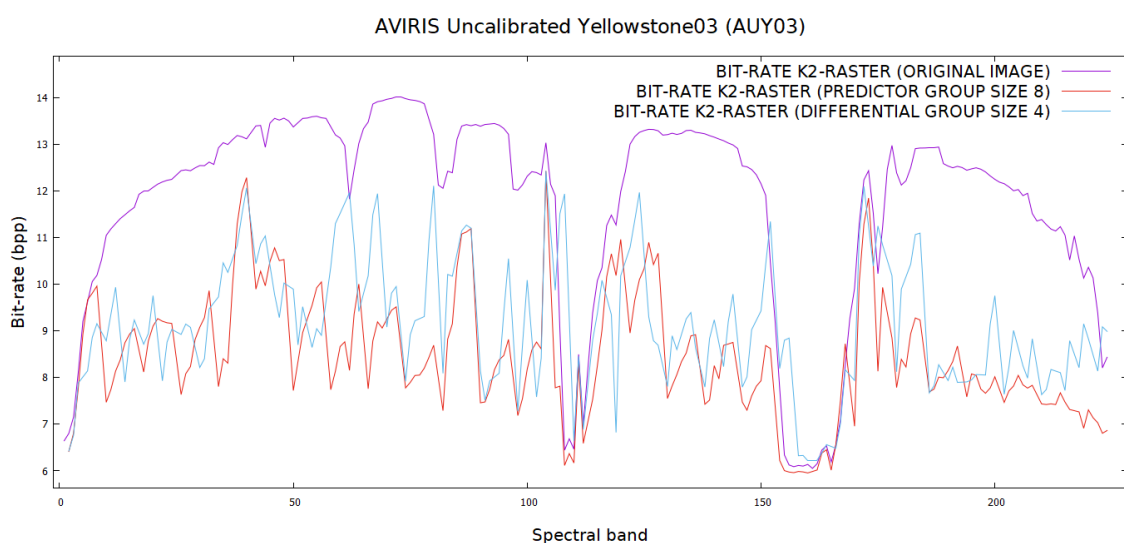
The proposed differential and predictive methods were used to transform these images into data with lower bit rates. They were then used as input to  $k^2$ -raster to further reduce their bit rates. Their performance was compared together with Reversible Haar Transform at levels 1 and 5, and the results are presented in Table 8. Figure 14 shows the entropy comparison of Yellowstone03 using differential and predictive methods while Figure 15 shows the bit rate comparison between the two

methods. Both show us that the proposed algorithm has brought benefits by lowering the entropy and the bit rates. The data for reference bands are left out of the plots so that the reader can have a clearer overall picture of the bit rate comparison.

Compared to other methods, the predictive method outperforms others, with the exception of Reversible Haar Transform level 5. However, it should be noted that while the predictive and differential methods require only two pixels (reference pixel and current pixel) to perform the reverse transformation, it would be a much more involved process to decode data using Reversible Haar Transform at a higher level. The experiments show that for all the testing images, the predictive method in almost all bands perform better than the differential method. This can be explained by the fact that in predictive encoding the values of  $\alpha$  and  $\beta$  in Equation (1) take into account not only the spectral correlation, but also the spatial correlation between the pixels in the bands when determining the prediction values. This is not the case with differential encoding whose values are only taken from the spectral correlation.



**Figure 14.** An entropy comparison of Yellowstone03 using differential and predictive methods. Data for reference bands are not included.



**Figure 15.** A bit rate comparison of Yellowstone03 using differential and predictive methods on  $k^2$ -raster. Data for reference bands are not included.



**Table 8.** A rate (bpbpb) comparison using different transformed methods: predictor, differential, reversible Haar level 1 and reversible Haar level 5 on  $k^2$ -raster. The optimal values are highlighted in red.

Hyperspectral Image	Transformation Type				
	Without Transformation	Predictor	Differential	Reversible Haar (Level 1)	Reversible Haar (Level 5)
AG9	9.49	6.76	7.52	8.10	6.83
AG16	9.12	6.63	7.29	7.81	6.60
ACY00	9.63	6.87	7.79	8.01	7.00
ACY03	9.44	6.72	7.65	7.86	6.87
AUY00	11.92	9.04	10.04	10.33	9.35
AUY03	11.74	8.87	9.91	10.18	9.23
C164	10.08	10.02	10.06	10.01	9.83
C165	10.37	10.33	10.37	10.33	10.16
HCA1	8.20	7.47	7.47	7.37	7.05
HCC	7.38	7.50	7.50	6.71	6.54
HFUEA	6.84	5.99	6.15	7.12	6.75
HFULM	6.79	6.06	6.18	7.14	6.83
I01	5.93	4.69	5.01	5.26	4.54
I02	5.90	4.75	5.03	5.26	4.57

#### 4. Conclusions

In this work, we have shown that using  $k^2$ -raster structure can help reduce the bit rates of a hyperspectral image. It also provides easy access to its elements without the need for initial full decompression. The predictive and differential methods can be applied to further reduce the rates. We performed experiments that showed that if the image data are first converted by either a predictive method or a differential method, we can gain more reduction in bit rates, thus making the storage capacity or the transmission volume of the data even smaller. The results of the experiments verified that the predictor indeed gives a better reduction in bit rates than the differential encoder and is preferred to be used for hyperspectral images.

For future work, we are interested in exploring the possibility of modifying the elements in a  $k^2$ -raster. This investigation is based on the dynamic structure,  $dk^2$ -tree, as discussed in the papers by de Bernardo et al. [29,30]. Additionally, we would like to improve on the variable-length encoding which is currently in use with  $k^2$ -raster, and hope to further reduce the size of the structure [23,24].

**Author Contributions:** Conceptualization, K.C., D.E.O.T., I.B. and J.S.-S.; methodology, K.C., D.E.O.T., I.B. and J.S.-S.; software, K.C.; validation, K.C., I.B. and J.S.-S.; formal analysis, K.C., D.E.O.T., I.B. and J.S.-S.; investigation, K.C., D.E.O.T., I.B. and J.S.-S.; resources, K.C., D.E.O.T., I.B. and J.S.-S.; data curation, K.C., I.B. and J.S.-S.; writing—original draft preparation, K.C., I.B. and J.S.-S.; writing—review and editing, K.C., I.B. and J.S.-S.; visualization, K.C., I.B. and J.S.-S.; supervision, I.B. and J.S.-S.; project administration, I.B. and J.S.-S.; funding acquisition, I.B. and J.S.-S.

**Funding:** This research was funded by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund under grants RTI2018-095287-B-I00 and TIN2015-71126-R (MINECO/FEDER, UE) and BES-2016-078369 (Programa Formación de Personal Investigador), and by the Catalan Government under grant 2017SGR-463.

**Acknowledgments:** The authors would like to thank Magli et al. for the M-CALIC software that they provided us in order to perform some of the experiments in this research work.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

AIRS	Atmospheric Infrared Sounder
AVIRIS	Airborne Visible InfraRed Imaging Spectrometer
CALIC	Context Adaptive Lossless Image Compression
CCSDS	Consultative Committee for Space Data Systems
CRISM	Compact Reconnaissance Imaging Spectrometer for Mars
DACs	Directly Addressable Codes
IASI	Infrared Atmospheric Sounding Interferometer
JPEG 2000	Joint Photographic Experts Group 2000
KLT	Karhunen–Loève Theorem
LOUDS	Level-Order Unary Degree Sequence
MDPI	Multidisciplinary Digital Publishing Institute
PCA	Principal Component Analysis
SOAP	Short Oligonucleotide Analysis Package

## References

1. Navarro, G. *Compact Data Structures: A Practical Approach*; Cambridge University Press: Cambridge, UK, 2016.
2. Jacobson, G. *Succinct Static Data Structures*. Ph.D. Thesis, Carnegie-Mellon, Pittsburgh, PA, USA, 1988.
3. Jacobson, G. Space-efficient static trees and graphs. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, Research Triangle Park, NC, USA, 30 October–1 November 1989; pp. 549–554.
4. Grossi, R.; Gupta, A.; Vitter, J.S. High-order entropy-compressed text indexes. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, Baltimore, MD, USA, 12–14 January 2003; Volume 72, pp. 841–850.
5. Ferragina, P.; Manzini, G. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, Redondo Beach, CA, USA, 12–14 November 2000; p. 390.
6. Burrows, M.; Wheeler, D. *A Block Sorting Lossless Data Compression Algorithm*; Technical Report; Digital Equipment Corporation: Maynard, MA, USA, 1994.
7. Langmead, B.; Trapnell, C.; Pop, M.; Salzberg, S.L. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.* **2009**, *10*, R25. [[CrossRef](#)] [[PubMed](#)]
8. Li, R.; Yu, C.; Li, Y.; Lam, T.; Yiu, S.; Kristiansen, K.; Wang, J. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics* **2009**, *25*, 1966–1967. [[CrossRef](#)] [[PubMed](#)]
9. Consultative Committee for Space Data Systems (CCSDS). *Image Data Compression CCSDS 123.0-B-1*; Blue Book; CCSDS: Washington, DC, USA, 2012.
10. Jolliffe, I.T. *Principal Component Analysis*; Springer: Berlin, Germany, 2002; p. 487.
11. Taubman, D.S.; Marcellin, M.W. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*; Kluwer Academic Publishers: Boston, MA, USA, 2001.
12. Wu, X.; Memon, N. CALIC—A context based adaptive lossless image CODEC. In *Proceedings of the 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, Atlanta, GA, USA, 9 May 1996.
13. Wu, X.; Memon, N. Context-based adaptive, lossless image coding. *IEEE Trans. Commun.* **1997**, *45*, 437–444. [[CrossRef](#)]
14. Wu, X.; Memon, N. Context-based lossless interband compression—Extending CALIC. *IEEE Trans. Image Process.* **2000**, *9*, 994–1001. [[PubMed](#)]
15. Magli, E.; Olmo, G.; Quacchio, E. Optimized onboard lossless and near-lossless compression of hyperspectral data using CALIC. *IEEE Geosci. Remote Sens. Lett.* **2004**, *1*, 21–25. [[CrossRef](#)]
16. Kiely, A.; Klimesh, M.; Blanes, I.; Ligo, J.; Magli, E.; Aranki, N.; Burl, M.; Camarero, R.; Cheng, M.; Dolinar, S.; et al. The new CCSDS standard for low-complexity lossless and near-lossless multispectral and hyperspectral image compression. In *Proceedings of the 6th International WS on On-Board Payload Data Compression (OBPDC)*, ESA/CNES, Matera, Italy, 20–21 September 2018.

17. Fjeldtvedt, J.; Orlandić, M.; Johansen, T.A. An efficient real-time FPGA implementation of the CCSDS-123 compression standard for hyperspectral images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2018**, *11*, 3841–3852. [[CrossRef](#)]
18. Báscones, D.; González, C.; Mozos, D. Hyperspectral image compression using vector quantization, PCA and JPEG2000. *Remote Sens.* **2018**, *10*, 907. [[CrossRef](#)]
19. Guerra, R.; Barrios, Y.; Díaz, M.; Santos, L.; López, S.; Sarmiento, R. A new algorithm for the on-board compression of hyperspectral images. *Remote Sens.* **2018**, *10*, 428. [[CrossRef](#)]
20. Ladra, S.; Paramá, J.R.; Silva-Coira, F. Scalable and queryable compressed storage structure for raster data. *Inf. Syst.* **2017**, *72*, 179–204. [[CrossRef](#)]
21. Samet, H. The Quadtree and related hierarchical data structures. *ACM Comput. Surv. (CSUR)* **1984**, *16*, 187–260. [[CrossRef](#)]
22. Brisaboa, N.R.; Ladra, S.; Navarro, G.  $k^2$ -trees for compact web graph representation. In *International Symposium on String Processing and Information Retrieval*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 18–30.
23. Brisaboa, N.R.; Ladra, S.; Navarro, G. DACs: Bringing direct access to variable-length codes. *Inf. Process Manag.* **2013**, *49*, 392–404. [[CrossRef](#)]
24. Brisaboa, N.R.; Ladra, S.; Navarro, G. Directly addressable variable-length codes. In *International Symposium on String Processing and Information Retrieval*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 122–130.
25. Silva-Coira, F. Compact Data Structures for Large and Complex Datasets. Ph.D. Thesis, Universidade da Coruña, A Coruña, Spain, 2017.
26. Cerdeira-Pena, A.; de Bernardo, G.; Fariña, A.; Paramá, J.R.; Silva-Coira, F. Towards a compact representation of temporal rasters. In *String Processing and Information Retrieval; SPIRE 2018; Lecture Notes in Computer Science*; Springer: Cham, Switzerland, 2018; Volume 11147.
27. Cruces, N.; Seco, D.; Gutiérrez, G. A compact representation of raster time series. In *Proceedings of the Data Compression Conference (DCC) 2019, Snowbird, UT, USA, 26–29 March 2019*; pp. 103–111.
28. Álvarez Cortés, S.; Serra-Sagristà, J.; Bartrina-Rapesta, J.; Marcellin, M. Regression Wavelet Analysis for Near-Lossless Remote Sensing Data Compression. *IEEE Trans. Geosci. Remote Sens.* **2019**. [[CrossRef](#)]
29. De Bernardo, G.; Álvarez García, S.; Brisaboa, N.R.; Navarro, G.; Pedreira, O. Compact queryable representations of raster data. In *International Symposium on String Processing and Information Retrieval*; Springer: Cham, Switzerland, 2013; pp. 96–108.
30. Brisaboa, N.R.; De Bernardo, G.; Navarro, G. Compressed dynamic binary relations. In *Proceedings of the 2012 Data Compression Conference, Snowbird, UT, USA, 10–12 April 2012*; pp. 52–61.






© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).

## Chapter 3

Analysis of variable-length codes  
for integer encoding in  
hyperspectral data compression  
with the  $k^2$ -raster compact data  
structure

Article

# Analysis of Variable-Length Codes for Integer Encoding in Hyperspectral Data Compression with the $k^2$ -Raster Compact Data Structure

Kevin Chow <sup>\*</sup>, Dion Eustathios Olivier Tzamarías, Miguel Hernández-Cabronero , Ian Blanes and Joan Serra-Sagristà 

Department of Information and Communications Engineering, Universitat Autònoma de Barcelona, 08193 Cerdanyola del Vallès, Barcelona, Spain; dion.tzamarías@uab.cat (D.E.O.T.); miguel.hernandez@uab.cat (M.H.-C.); ian.blanes@uab.cat (I.B.); joan.serra@uab.cat (J.S.-S.)

\* Correspondence: kevin.chow@uab.cat

Received: 19 May 2020; Accepted: 18 June 2020; Published: 20 June 2020



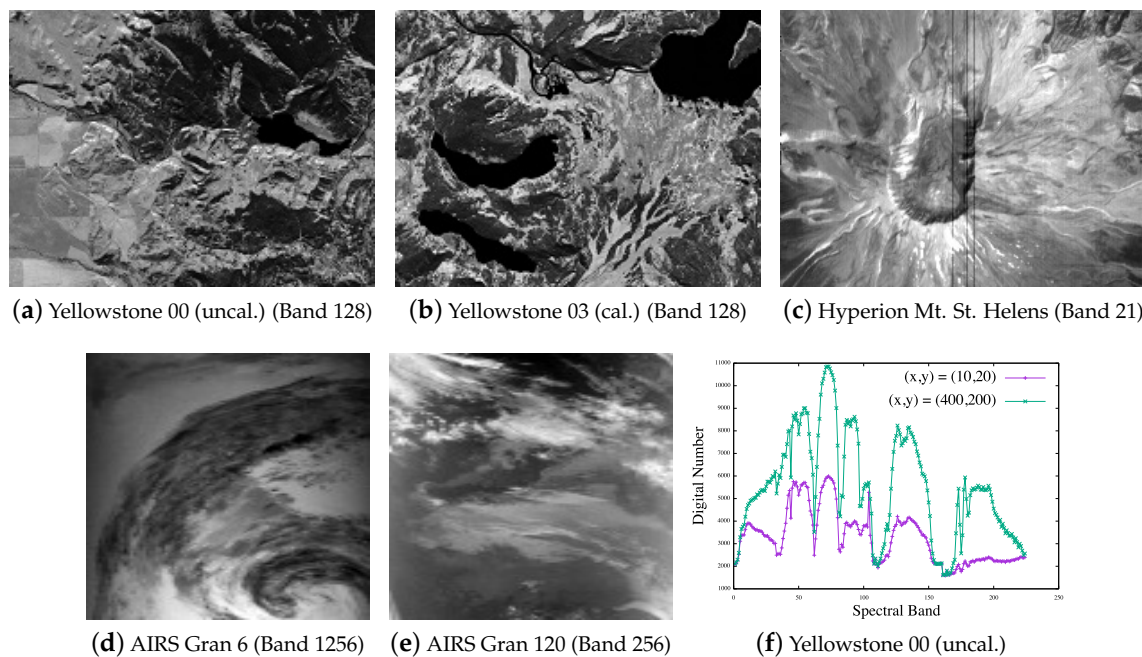
**Abstract:** This paper examines the various variable-length encoders that provide integer encoding to hyperspectral scene data within a  $k^2$ -raster compact data structure. This compact data structure leads to a compression ratio similar to that produced by some of the classical compression techniques. This compact data structure also provides direct access for query to its data elements without requiring any decompression. The selection of the integer encoder is critical for obtaining a competitive performance considering both the compression ratio and access time. In this research, we show experimental results of different integer encoders such as Rice, Simple9, Simple16, PForDelta codes, and DACs. Further, a method to determine an appropriate  $k$  value for building a  $k^2$ -raster compact data structure with competitive performance is discussed.

**Keywords:** compact data structure;  $k^2$ -raster; DACs; Elias codes; Simple9; Simple16; PForDelta; Rice codes; hyperspectral scenes

## 1. Introduction

Hyperspectral scenes [1–10] are data taken from the air by sensors such as AVIRIS (Airborne Visible/Infrared Imaging Spectrometer) or by satellite instruments such as Hyperion and IASI (Infrared Atmospheric Sounding Interferometer). These scenes are made up of multiple bands from across the electromagnetic spectrum, and data extracted from certain bands are helpful in finding objects such as oil fields [11] or minerals [12]. Other applications include weather prediction [13] and wildfire soil studies [14], to name a few. Due to their sizes, hyperspectral scenes are usually compressed to facilitate their transmission and reduce storage size.

Compact data structures [15] are a type of data structure where data are stored efficiently while at the same time providing real-time processing and compression of the data. They can be loaded into main memory and accessed directly by means of the rank and select functions [16] in the structures. Compressed data provide reduced space usage and query time, i.e., they allow more efficient transmission through limited communication channels, as well as faster data access. There is no need to decompress a large portion of the structure to access and query individual data as is the case with data compressed by classical compression algorithms such as gzip or bzip2 and by specialized algorithms such as CCSDS123.0-B-1 [17] or KLT+JPEG 2000 [18,19]. In this paper, we are interested in lossless compression of hyperspectral scenes through compact data structures. Therefore, reconstructed scenes should be identical to the originals before compression. Any deterministic analysis process will necessarily yield the same results. Figure 1 shows several images from our datasets.



**Figure 1.** Several hyperspectral scenes used in this paper. The original and the decompressed scenes discussed in this paper are numerically identical. Depicted also are two spectral signatures for AVIRIS Yellowstone 00 (uncal.). The AVIRIS images in this figure are courtesy of NASA/JPL-Caltech.

The compact data structure used in this paper is called  $k^2$ -raster. It is a tree structure developed from another compact data structure called  $k^2$ -tree.  $k^2$ -raster is built from a raster matrix with its pixel cells filled with integer values, while  $k^2$ -tree is from a bitmap matrix with zero and one values. During the construction of the  $k^2$ -raster tree, if the neighboring pixels have equal values such as clusters (spatial correlation), the number of nodes in the tree that need to be saved is reduced. If the values are similar, as discussed later in this paper, the values will be made even smaller. They are then compressed or packed in a more compact form by the integer encoders, and with these small integers, the compression results are even better. Moreover, when it comes to querying cells, a tree structure speeds up the search, saving access time. Another added advantage of some of the integer encoders is that they provide direct random access to the cells without any need for full decompression.

Currently, huge amounts of remote sensing data have been produced, transmitted, and archived, and we can foresee that in the future, the amount of larger datasets is expected to keep growing at a fast rate. The need for their compression is becoming more pressing and critical. In view of this trend, we take on the task of remote sensing compression and make it as one of our main objectives. In this research work, we reduce hyperspectral data sizes by using compact data structures to produce lossless compression. Early on, we began by examining the possibility of taking advantage of the spatial correlation and spectral correlation in the data. In our previous paper [20], we presented a predictive method and a differential method that made use of these correlations in hyperspectral data with favorable results. However, in this paper, we would like to focus on selecting a suitable integer encoder that is employed in the  $k^2$ -raster compact data structure, as that is also a major factor in providing competitive compression ratios.

Compression of integer data in the most effective and efficient way, in relation to compact data structures, has been the focus of many studies over the past several decades. Some include Elias [21–23], Rice [24–26], PForDelta [27–29], and Directly Addressable Codes (DACs) [30–32]. In our case, we need to store non-negative, typically small integers in the  $k^2$ -raster structure. This structure is a tree built in such a way that the nodes are not connected by pointers, but can still be reached with the use of a compact data structure linear rank function. When the data are saved, no pointers need to be stored, thus keeping the size of the structure small. Additionally, we use a fixed code ([15], §2.7)



to help us save even more space. In what follows, we investigate the effectiveness of some of these integer encoders.

The rest of the paper is organized as follows: In Section 2, we describe the  $k^2$ -raster structure, followed by the various variable-length integer encoders such as Elias, Rice, PForDelta, and DACs. Section 3 presents experimental results for finding the best and optimal values for  $k$  and exploring the different integer encoders for  $k^2$ -raster. This is done in comparison with classical compression techniques. Lastly, some conclusions and final thoughts on future work are put forth in Section 4.

## 2. Materials and Methods

In this section, we describe  $k^2$ -raster [33] and the integer encoders Elias, Rice, Simple9, Simple16, PForDelta codes, and DACs. This is followed by a discussion on how to obtain the best value of  $k$  and two related works on raster compression: heuristic  $k^2$ -raster [33] and 3D-2D mapping [34].

### 2.1. $K^2$ -Raster

The  $k^2$ -tree structure was originally proposed by Ladra et al. [35] as a compact representation of the adjacency matrix of a directed graph. Its applications include web graphs and social networks. Based on  $k^2$ -tree, the same authors also proposed  $k^2$ -raster [33], which is specifically designed for raster data including images. A  $k^2$ -raster is built from a matrix of width  $w$  and height  $h$ . If the matrix can be partitioned into  $k^2$  square subquadrants of equal size, it can be used directly. Otherwise, it is necessary to enlarge the matrix to size  $s \times s$ , where  $s$  is computed as:

$$s = k^{\lceil \log_k \max(w, h) \rceil}, \quad (1)$$

setting the new elements to 0. This extended matrix is then recursively partitioned into  $k^2$  submatrices of identical size, referred to as quadrants. This process is repeated until all cells in a quadrant have the same value, or until the submatrix has size  $1 \times 1$  and cannot be further subdivided. This partitioning induces a tree topology, which is represented in a bitmap  $T$ . Elements can then be accessed via a rank function. At each tree level, the maximum and minimum values of each quadrant are computed. These are then compared with the corresponding maximum and minimum values of the parent, and the differences are stored in the  $Vmax$  and  $Vmin$  arrays of each level. Saving the differences instead of the original values results in lower values for each node, which in turn allows a better compression with DACs or other integer encoders such as Simple9, PForDelta, etc. An example of a simple  $8 \times 8$  matrix is given in Figure 2 to illustrate this process. A  $k^2$ -raster is constructed from this matrix with maximum and minimum values as given in Figure 3. Differences from the parents' extrema are then computed as explained above, resulting in the structure shown in Figure 4. Next, with the exception of the root node at the top level, the  $Vmax$  and  $Vmin$  arrays at all levels are concatenated to form  $Lmax$  and  $Lmin$ , respectively. Both arrays are then compressed by an integer encoder such as DACs. The root's maximum ( $rMax$ ) and minimum ( $rMin$ ) values remain uncompressed. The resulting elements, which fully describe this  $k^2$ -raster structure, are given in Table 1.

### 2.2. Unary Codes and Notation

We denote  $x$  as a non-negative integer. The expression  $|x|$  gives the minimum bit length needed to express  $x$ , i.e.,  $|x| = \lfloor \log_2 x \rfloor + 1$ .

Unary codes are generally used for small integers. Unary codes have the following form:

$$u(x) = 0^x 1, \quad (2)$$

where the superscript  $x$  indicates the number of consecutive 0 bits in the code. For example,  $u(1_d) = 0^1 1 = 01_b$ ,  $u(6_d) = 0^6 1 = 000001_b$ ,  $u(9_d) = 0^9 1 = 000000001_b$ . Here, bits are denoted by a subscript  $b$  and decimal numbers by a subscript  $d$ . Furthermore, when codes are composed of two parts, they are spaced apart for readability purposes. In general, the notation used in [15] is adopted in this paper.

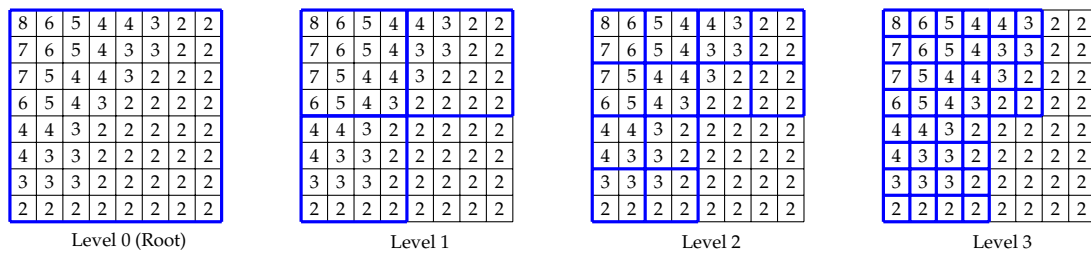


Figure 2. Subdivision of an example  $8 \times 8$  matrix for  $k^2$ -raster ( $k = 2$ ).

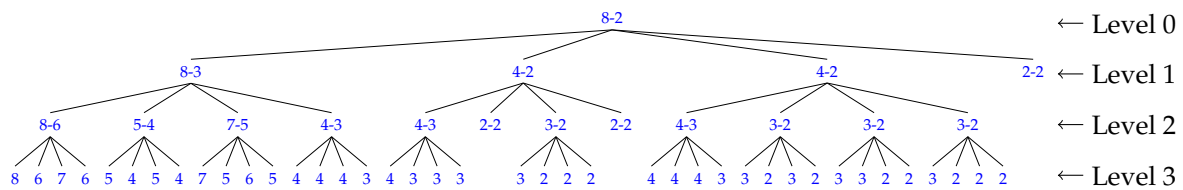


Figure 3. A  $k^2$ -raster ( $k = 2$ ) tree storing the maximum and minimum values for each quadrant of every recursive subdivision of the matrix in Figure 2. Every node contains the maximum and minimum values of the subquadrant, separated by a dash. On the last level, only one value is shown as each subquadrant contains only one cell.

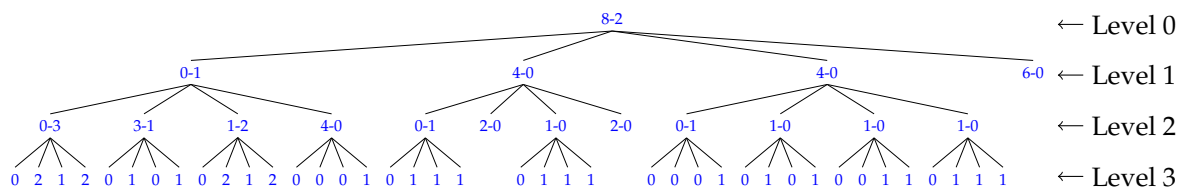


Figure 4. Based on the tree in Figure 3, the maximum value of each node is subtracted from that of its parent while the minimum value of the parent is subtracted from the node’s minimum value. These differences then replace their corresponding values in the node. The maximum and minimum values of the root remain the same.

Table 1. An example of the elements of a  $k^2$ -raster based on Figures 2–4.

$T$ Bitmap	binary	1110 1111 1010 1111
$L_{max}$	decimal	Level 1 0446
		Level 2 0314 0212 0111
		Level 3 0212 0101 0212 0001 0111 0111 0001 0101 0011 0111
$L_{min}$	decimal	Level 1 100
		Level 2 3120 10 1000
$rMax$	decimal	8
$rMin$	decimal	2

### 2.3. Elias Codes

Elias codes include Gamma ( $\gamma$ ) codes and Delta ( $\delta$ ) codes. They were developed by Peter Elias [21] to encode natural numbers, and in general, they work well with sequences of small numbers.

Gamma codes have the following form:

$$\gamma(x) = 0^{|x|-1} [x]_{|x|} = u(|x| - 1) [x]_{|x|-1}, \tag{3}$$



where  $[x]_l$  represents the  $l$  least significant bits of  $x$ . For example,  $\gamma(1_d) = \gamma(1_b) = 1_b$ ,  $\gamma(4_d) = \gamma(100_b) = 001\ 00_b$ ,  $\gamma(6_d) = \gamma(110_b) = 001\ 10_b$ ,  $\gamma(9_d) = \gamma(1001_b) = 0001\ 001_b$ ,  $\gamma(14_d) = \gamma(1110_b) = 0001\ 110_b$ .

Delta codes have the following form:

$$\delta(x) = \gamma(|x|) [x]_{|x|-1} \tag{4}$$

For values that are larger than 31, Delta codes produce shorter codewords than Gamma codes. This is due to the use of Gamma codes in forming the first part of their codes, which provides a shorter code length for Delta codes as the number becomes larger. Some examples are:  $\delta(1_d) = \delta(1_b) = 1_b$ ,  $\delta(6_d) = \delta(110_b) = 011\ 10_b$ ,  $\delta(9_d) = \delta(1001_b) = 00100\ 001_b$ ,  $\delta(14_d) = \delta(1110_b) = 00100\ 110_b$ .

#### 2.4. Rice Codes

Rice codes [25] are a special case of Golomb codes. Let  $x$  be an integer value in the sequence, and let  $y = \lfloor x/2^l \rfloor$ , where  $l$  is a non-negative integer parameter. The Rice codes for this parameter are defined as:

$$R_l(x) = u(y + 1) [x]_l \tag{5}$$

Some examples are shown for different values of  $l$  in Table 2.

Table 2. Some examples of Rice codes.

Value $v$		Rice Code $R_l(v)$			
Decimal	Binary	$l = 1$	$l = 2$	$l = 3$	$l = 4$
$1_d$	$1_b$	$1_b$	$1_b$	$1_b$	$1_b$
$6_d$	$110_b$	$0001\ 0_b$	$01\ 10_b$	$110_b$	$110_b$
$9_d$	$1001_b$	$00001\ 1_b$	$001\ 01_b$	$01\ 001_b$	$1001_b$
$14_d$	$1110_b$	$00000001\ 0_b$	$0001\ 10_b$	$01\ 110_b$	$1110_b$

To obtain optimal performance among Rice codes,  $l$  should be selected to be close to the expected value of the input integers. In general, Rice codes give better compression performance than Elias  $\gamma$  and  $\delta$  codes.

#### 2.5. Simple9, Simple16, and PForDelta

Apart from Elias codes and Rice codes, the codes in this section store the integers in single or multiple word-sized elements to achieve data compression. They have been shown to have good compression ratios [30].

Simple9 [36] assigns a maximum possible number of a certain bit length to a 28-bit segment or packing space of a 32-bit word. The other 4 bits contain a selector that has a value ranging from 0 to 8. Each selector has information that indicates how the integers are stored, and that includes the number of these integers and the maximum number of bits that each integer is allowed in this packing space. For example, Selector 0 tests to see if the first 28 integers in the data have a value of 0 or 1, i.e., a bit length of 1. If they do, then they are stored in this 28-bit segment. Otherwise, Selector 1 tests to see if it can pack 14 integers into the segment with a maximum bit length of 2 bits for each. If this still does not work, Selector 2 tests to see if 9 integers can each be packed into a maximum bit length of 3 bits. This testing goes on until the right number of data are found that can be stored in these 28 bits. Table 3 shows the 9 different ways of using 28 bits in a word of 32 bits in Simple9.

Simple16 [37] is a variant of Simple9 and uses all 16 combinations in the selector bits. Their values range from 0 to 15. Table 4 shows the 16 different ways of packing integers into the 28-bit segment in Simple16.

PForDelta [27] is also similar to both Simple9 and Simple16, but encodes a fixed group of numbers at a time. To do so, 128- or 256-bit words are used.

Due to its relative simplicity, Simple9 is used here as an example to illustrate how an integer sequence is stored in the encoders described in this section. This sequence  $\langle 3591\ 25\ 13\ 12\ 15\ 12\ 11\ 26\ 20\ 8\ 13\ 8\ 9\ 7\ 13\ 10\ 12\ 0\ 10 \rangle_d$  is taken from the  $L_{max}$  array of one of our data scenes AG9, and the bit-packing is shown in Table 5. There are 19 integers in the sequence. Assuming the integer is 16 bits each, the sequence has a total size of 38 bytes. After packing into the array, the sequence occupies only 16 bytes.

**Table 3.** Nine different ways of encoding numbers in the 28-bit packing space in Simple9.

Selector	Number of Integers $n$	Width of Integers $\lfloor 28/n \rfloor$ (Bits)	Wasted Bits
0	28	1	0
1	14	2	0
2	9	3	1
3	7	4	0
4	5	5	3
5	4	7	0
6	3	9	1
7	2	14	0
8	1	28	0

**Table 4.** Sixteen different ways of encoding numbers in the 28-bit packing space in Simple16. There are no wasted bits in any of the selectors.

Selector	Number of Integers	Width of Integers (Bits)
0	28	$28 \times 1$ bit
1	21	$7 \times 2$ bits, $14 \times 1$ bit
2	21	$7 \times 1$ bit, $7 \times 2$ bits, $7 \times 1$ bit
3	21	$14 \times 1$ bit, $7 \times 2$ bits
4	14	$14 \times 2$ bits
5	9	$1 \times 4$ bits, $8 \times 3$ bits
6	8	$1 \times 3$ bits, $4 \times 4$ bits, $3 \times 3$ bits
7	7	$7 \times 4$ bits
8	6	$4 \times 5$ bits, $2 \times 4$ bits
9	6	$2 \times 4$ bits, $4 \times 5$ bits
10	5	$3 \times 6$ bits, $2 \times 5$ bits
11	5	$2 \times 5$ bits, $3 \times 6$ bits
12	4	$4 \times 7$ bits
13	3	$1 \times 10$ bits, $2 \times 9$ bits
14	2	$2 \times 14$ bits
15	1	$1 \times 28$ bits

**Table 5.** Example to show how the integer sequence  $\langle 3591\ 25\ 13\ 12\ 15\ 12\ 11\ 26\ 20\ 8\ 13\ 8\ 9\ 7\ 13\ 10\ 12\ 0\ 10 \rangle_d$  is stored with Simple9.

Element	Selector	Number of Integers	Integers Stored (Decimal)	Integers Stored (Binary)
0	$7_d (0111_b)$	2	3591 25	00111000000111 00000000011001
1	$4_d (0100_b)$	5	13 12 15 12 11	01101 01100 01111 01100 01011
2	$4_d (0100_b)$	5	26 20 8 13 8	11010 10100 01000 01101 01000
3	$3_d (0011_b)$	7	9 7 13 10 12 0 10	1001 0111 1101 1010 1100 0000 1010

### 2.6. Directly Addressable Codes

Directly Addressable Codes (DACs) can be used to compress  $k^2$ -raster and provide access to variable-length codes. Based on the concept of compact data structures, DACs were proposed in the papers published by Brisaboa et al. in 2009 [30] and 2013 [31]. This structure is proven to yield good compression ratios for variable-length integer sequences. By means of the rank function, it gains fast direct access to any position of the sequence in a very compact space. The original authors also asserted that it was best suited for a sequence of integers with a skewed frequency distribution toward smaller integer values.

Different types of encoding are used for DACs, and the one that we are interested in for  $k^2$ -raster is called VBytecoding. Consider a sequence of integers  $x$ . Each integer  $x_i$ , which is represented by  $\lfloor \log_2 x_i \rfloor + 1$  bits, is broken into chunks of bits of size  $C_S$ . Each chunk is stored in a block of size  $C_S + 1$  with the additional bit used as a control bit. The chunk occupies the lower bits in the block and the control bit the highest bit. The block that holds the most significant bits of the integer has its control bit set to 0, while the others have it set to 1. For example, if we have an integer  $41_d$  ( $101001_b$ ), which is 6 bits long, and if the chunk size is  $C_S = 3$ , then we have 2 blocks:  $\underline{0}101 \underline{1}001_b$ . The control bit in each block is shown underlined. To show how the blocks are organized and stored, we again illustrate it with an example. Given five integers of variable length:  $7_d$  ( $111_b$ ),  $41_d$  ( $101001_b$ ),  $100_d$  ( $1100100_b$ ),  $63_d$  ( $111111_b$ ),  $427_d$  ( $110101011_b$ ), and a chunk size of 3 (the block size is 4), their representations are listed in Table 6.

**Table 6.** Example of an integer sequence and the corresponding DACs blocks of the integers.

Decimal	Binary	DACs Blocks
$7_d$	$\underline{0}111_b$	$(B_{1,1}A_{1,1})$
$41_d$	$\underline{0}101 \underline{1}001_b$	$(B_{2,2}A_{2,2} B_{2,1}A_{2,1})$
$100_d$	$\underline{0}001 \underline{1}100 \underline{1}100_b$	$(B_{3,3}A_{3,3} B_{3,2}A_{3,2} B_{3,1}A_{3,1})$
$63_d$	$\underline{0}111 \underline{1}111_b$	$(B_{4,2}A_{4,2} B_{4,1}A_{4,1})$
$427_d$	$\underline{0}110 \underline{1}101 \underline{1}011_b$	$(B_{5,3}A_{5,3} B_{5,2}A_{5,2} B_{5,1}A_{5,1})$

We store them in three blocks of arrays  $A$  and control bitmaps  $B$ . This is depicted in Figure 5. To retrieve the values in the arrays  $A$ , we make use of the corresponding bitmaps  $B$  with the rank function. This function returns the number of bits, which are set to 1 from the beginning position to the one being queried in the control bitmap  $B_i$ . An example of how the function is used follows: If we want to access the third integer ( $100_d$ ) in the sequence in Figure 5, we start looking for the third element in the array  $A_1$  in Block<sub>1</sub> and find  $A_{3,1}$  with its corresponding control bitmap  $B_{3,1}$ . The function  $\text{rank}(B_{3,1})$  then gives a result of 2, which means that the second element  $A_{3,2}$  in the array  $A_2$  in Block<sub>2</sub> contains the next block. With the control bit in  $B_{3,2}$ , we compute the function  $\text{rank}(B_{3,2})$  and obtain a result of 1. This means the next block in Block<sub>3</sub> can be found in the first element  $A_{3,3}$ . Since its corresponding control bitmap  $B_{3,3}$  is set to 0, the search ends here. All the blocks found are finally concatenated to form the third integer in the sequence.

More information on DACs and the software code can be found in the papers [30,31] by Ladra et al.

Block <sub>1</sub>	$A_1$	111 ( $A_{1,1}$ )	001 ( $A_{2,1}$ )	100 ( $A_{3,1}$ )	111 ( $A_{4,1}$ )	011 ( $A_{5,1}$ )
	$B_1$	0 ( $B_{1,1}$ )	1 ( $B_{2,1}$ )	1 ( $B_{3,1}$ )	1 ( $B_{4,1}$ )	1 ( $B_{5,1}$ )
Block <sub>2</sub>	$A_2$	101 ( $A_{2,2}$ )	100 ( $A_{3,2}$ )	111 ( $A_{4,2}$ )	101 ( $A_{5,2}$ )	
	$B_2$	0 ( $B_{2,2}$ )	1 ( $B_{3,2}$ )	0 ( $B_{4,2}$ )	1 ( $B_{5,2}$ )	
Block <sub>3</sub>	$A_3$	001 ( $A_{3,3}$ )	110 ( $A_{5,3}$ )			
	$B_3$	0 ( $B_{3,3}$ )	0 ( $B_{5,3}$ )			

**Figure 5.** Organization of 3 DACs blocks.

### 2.7. Selection of the $k$ Value

Following the description of Subsection 2.1, using different  $k$  values leads to the creation of  $L_{max}$  and  $L_{min}$  arrays of different lengths. This, in turn, affects the final results of the size of  $k^2$ -raster. With this in mind, we present a heuristic approach that can be used to determine the best  $k$  value for obtaining the smallest storage size. First, we compute the sizes of the extended matrix for different values of  $k$  within a suitable range using Equation (1). Then, we find the  $k$  value that corresponds to

the matrix with the smallest size, and the result can be considered as the best  $k$  value. Before the start of the  $k^2$ -raster building process, the program can find the best  $k$  value and use it as the default.

### 2.8. Heuristic $k^2$ -Raster

In the  $k^2$ -raster paper by Ladra et al. [33], a variant of this structure was also proposed whereby the elements at the last level of the tree structure are stored by using an entropy-based heuristic approach. This is denoted by  $k_H^2$ -raster. For example, for  $k = 2$ , each set of the 4 nodes that are from the same parent forms a codeword. It is possible that at this same level of the tree, these codewords may be repeated, and their frequencies of occurrences can be computed. These sets of codewords and their frequencies are then compressed and saved. In effect, the more these codewords are repeated, the less storage space they take up. An example of codeword frequency based on the  $k^2$ -raster discussed in Section 2.1 is shown in Table 7. According to experiments conducted by the authors of [33], it saves space in the final representation.

**Table 7.** Codeword frequency in Level 3 of the  $Lmax$  bitmap in the  $k^2$ -raster structure in Figure 1.

Codeword	Frequency
0111	3
0212	2
0101	2
0001	2
0011	1

### 2.9. 3D-2D Mapping

A study on compact representation of raster images in a time-series was proposed by Cruces et al. in [34]. This method is based on the 3D to 2D mapping of a raster where 3D tuples  $\langle x, y, z \rangle$  are mapped into a 2D binary grid. That is, a raster of size  $w \times h$  with values in a certain range, between 0 and  $v$  inclusive, has a binary matrix of  $w \times h$  columns and  $v+1$  rows. All the rasters are then concatenated into a 3D matrix and stored as a 3D- $k^2$ -tree.

## 3. Experimental Results

In this section, we present an exhaustive comparison of the different integer encoders for use with  $k^2$ -raster. First, though, we report results from experiments for finding the best  $k$  value. Reported also are the experimental results to find out if the heuristic  $k^2$ -raster and 3D-2D mapping would give better storage sizes. All storage sizes in this section are expressed as bits per pixel per band (bpppb).

The hyperspectral scenes were captured by different sensors: Atmospheric Infrared Sounder (AIRS), AVIRIS, Compact Reconnaissance Imaging Spectrometer for Mars (CRISM), Hyperion, and IASI. Except for IASI, all of them are publicly available for download (<http://cwe.ccsds.org/sls/docs/sls-dc/123.0-B-Info/TestData>). The hyperspectral scenes used are listed in Table 8.

The implementations for  $k^2$ -raster and  $k_H^2$ -raster were based on the algorithms presented in the paper by Ladra et al. [33]. The sds-lite implementation of  $k^2$ -tree by Simon Gog [38] ([https://github.com/simongog/sds-lite/blob/master/include/sdsl/k2\\_tree.hpp](https://github.com/simongog/sds-lite/blob/master/include/sdsl/k2_tree.hpp)) was used for testing 3D-2D mapping described in the paper by Cruces et al. [34]. The DACs software was downloaded from a package called “DACs, optimization with no further restrictions” at the Universidade da Coruña’s Database Laboratory website (<http://lbd.udc.es/research/DACS/>). The programming code for the Rice, PForDelta, Simple9, and Simple16 codes was written by the programmers Diego Caro, Michael Dipperstein, and Christopher Hoobin and was downloaded from these authors’ GitHub web pages. Slight modifications to the code were made to meet our requirements to perform the experiments. All programs for this paper were written in C and C++ and compiled with gnu g++ 5.4.0 20160609 with -Ofast optimization. The experiments were carried out on an Intel Core 2 Duo

CPU E7400 @2.80GHz with 3072KB of cache and 3GB of RAM. The operating system was Ubuntu 16.04.5 LTS with kernel 4.15.0-47-generic (64 bits). The software code is available at <http://gici.uab.cat/GiciWebPage/downloads.php>.

**Table 8.** Hyperspectral scenes used in our experiments. Also shown are the bit rate and bit rate reduction using  $k^2$ -raster.  $x$  is the scene width,  $y$  the scene height, and  $z$  the number of spectral bands. bpppb, bits per pixel per band; CRISM, Compact Reconnaissance Imaging Spectrometer for Mars; IASI, Infrared Atmospheric Sounding Interferometer.

Sensor	Name	C/U *	Acronym	Original Dimensions ( $x \times y \times z$ )	Bit Depth (bpppb)	Best $k$ Value	$k^2$ -raster Bit Rate (bpppb)	$k^2$ -raster Bit-Rate Reduction (%)	
AIRS	9	U	AG9	$90 \times 135 \times 1501$	12	6	9.49	21%	
	16	U	AG16	$90 \times 135 \times 1501$	12	6	9.12	24%	
	60	U	AG60	$90 \times 135 \times 1501$	12	15	9.72	19%	
	126	U	AG126	$90 \times 135 \times 1501$	12	6	9.61	20%	
	129	U	AG129	$90 \times 135 \times 1501$	12	6	8.65	28%	
	151	U	AG151	$90 \times 135 \times 1501$	12	6	9.53	21%	
	182	U	AG182	$90 \times 135 \times 1501$	12	6	9.68	19%	
	193	U	AG193	$90 \times 135 \times 1501$	12	15	9.30	23%	
AVIRIS	Yellowstone sc. 00	C	ACY00	$677 \times 512 \times 224$	16	6	9.61	40%	
	Yellowstone sc. 03	C	ACY03	$677 \times 512 \times 224$	16	6	9.42	41%	
	Yellowstone sc. 10	C	ACY10	$677 \times 512 \times 224$	16	6	7.62	52%	
	Yellowstone sc. 11	C	ACY11	$677 \times 512 \times 224$	16	6	8.81	45%	
	Yellowstone sc. 18	C	ACY18	$677 \times 512 \times 224$	16	6	9.78	39%	
	Yellowstone sc. 00	U	AUY00	$680 \times 512 \times 224$	16	9	11.92	25%	
	Yellowstone sc. 03	U	AUY03	$680 \times 512 \times 224$	16	9	11.74	27%	
	Yellowstone sc. 10	U	AUY10	$680 \times 512 \times 224$	16	9	9.99	38%	
	Yellowstone sc. 11	U	AUY11	$680 \times 512 \times 224$	16	9	11.27	30%	
	Yellowstone sc. 18	U	AUY18	$680 \times 512 \times 224$	16	9	12.15	24%	
	CRISM	frt000065e6_07_sc164	U	C164	$640 \times 420 \times 545$	12	6	10.08	16%
		frt00008849_07_sc165	U	C165	$640 \times 450 \times 545$	12	6	10.37	14%
		frt0001077d_07_sc166	U	C166	$640 \times 480 \times 545$	12	6	11.05	8%
		hrl00004f38_07_sc181	U	C181	$320 \times 420 \times 545$	12	5	9.97	17%
hrl0000648f_07_sc182		U	C182	$320 \times 450 \times 545$	12	5	10.11	16%	
hrl0000ba9c_07_sc183		U	C183	$320 \times 480 \times 545$	12	5	10.65	11%	
Hyperion	Agricultural	C	HCA	$256 \times 3129 \times 242$	12	16	8.52	29%	
	Coral Reef	C	HCC	$256 \times 3127 \times 242$	12	8	7.62	36%	
	Urban	C	HCU	$256 \times 2905 \times 242$	12	16	8.85	26%	
	Erta Ale	U	HUEA	$256 \times 3187 \times 242$	12	8	7.76	35%	
	Lake Monona	U	HULM	$256 \times 3176 \times 242$	12	8	7.82	35%	
	Mt. St. Helena	U	HUMS	$256 \times 3242 \times 242$	12	8	7.91	34%	
IASI	Level 0 1	U	I01	$60 \times 1528 \times 8359$	12	12	6.32	47%	
	Level 0 2	U	I02	$60 \times 1528 \times 8359$	12	12	6.38	47%	
	Level 0 3	U	I03	$60 \times 1528 \times 8359$	12	12	6.31	47%	
	Level 0 4	U	I04	$60 \times 1528 \times 8359$	12	12	6.43	46%	

\*: Calibrated (C) or Uncalibrated (U).

### 3.1. Best $k$ Value Selection

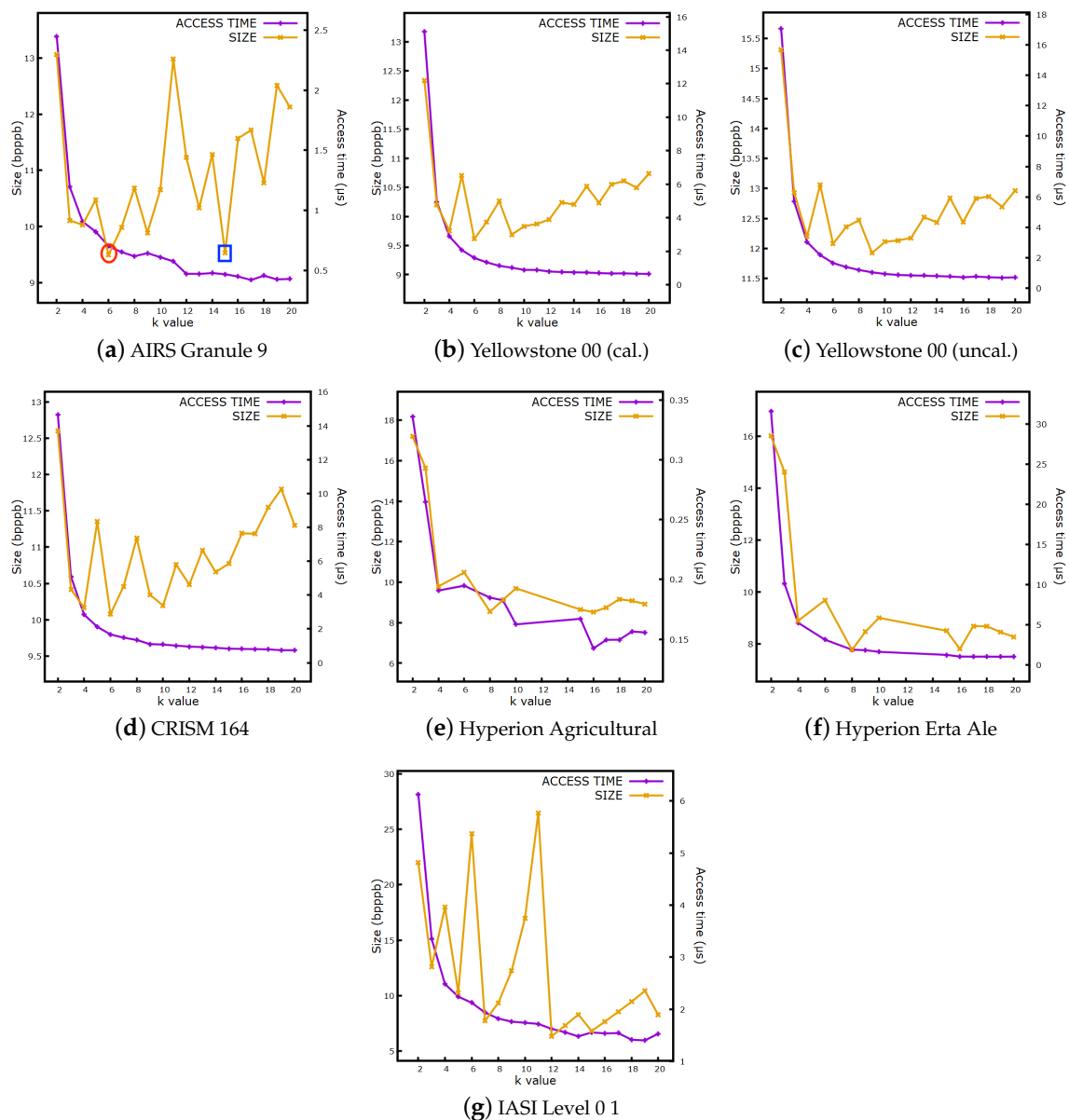
From our previous research [20], the selection of the  $k$  value when building a  $k^2$ -raster was shown to have a great effect on the resulting size of the structure, as well as the access time to query its elements. In order to further investigate this idea, we extended our research to finding ways of choosing the best  $k$  value. One way was to build the  $k^2$ -raster structure with different  $k$  values for scene data from each sensor to see how the matrix size affected the choice of the  $k$  value. Additionally, we measured the time it took to build the  $k^2$ -raster and the size of the structure. The results are shown in Table 9. For most tested data, the  $k$  value leading to the smallest extended matrix size (attribute S in the table) usually provided the fastest build time and the smallest storage size. With these results, we could say that, in general, when  $k = 2$ , the compressed data size was large, sometimes even larger than the size of the original scene. As the value of  $k$  became larger, beginning with  $k = 3$ , the compressed data size was reduced. As far as the compressed size was concerned, the best value was in the range from three to 10 for matrices with a small raster size (i.e., if both the original width and original height were less than 1000) such as the ones for the AIRS Granule or AVIRIS Yellowstone scenes. If at least one dimension was larger than 1000 such as Hyperion calibrated or uncalibrated scenes, a larger range, typically between three and 20, needed to be considered.

**Table 9.** Results for different  $k$  values using the scene data from each sensor for the following attributes: (S) the extended matrix Size (pixels), (C) the  $k^2$ -raster Compressed storage data rate (bpppb), and (B) the time to Build the  $k^2$ -raster (seconds). The original scene width and height are shown in the first column. The best results are highlighted in blue.

Scene Data (w × h) *		k=2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AG9 (90 × 135)	S	256	243	256	625	216	343	512	729	1000	1331	144	169	196	225	256	289	324	361	400
	C	13.06	10.11	10.03	10.47	9.49	9.98	10.68	9.89	10.65	12.98	11.23	10.33	11.29	9.53	11.57	11.72	10.78	12.52	12.13
	B	5.3	3.2	4.1	10.9	4.2	10.9	12.6	10.7	17.5	29.6	2.9	4.1	3.0	4.3	4.6	6.6	6.8	4.9	7.3
ACY00 (677 × 512)	S	1024	729	1024	3125	1296	2401	4096	729	1000	1331	1728	2197	2744	3375	4096	4913	5832	6859	8000
	C	12.34	10.20	9.76	10.70	9.61	9.91	10.26	9.69	9.83	9.87	9.95	10.24	10.20	10.51	10.24	10.55	10.61	10.49	10.73
	B	19.5	10.7	10.8	30.7	10.5	19.3	42.0	8.8	9.3	11.5	13.2	17.1	23.2	29.1	45.5	55.8	72.6	101.1	131.0
AUY00 (680 × 512)	S	1024	729	1024	3125	1296	2401	4096	729	1000	1331	1728	2197	2744	3375	4096	4913	5832	6859	8000
	C	15.31	12.93	12.20	13.06	12.08	12.35	12.47	11.92	12.11	12.13	12.17	12.52	12.43	12.84	12.44	12.83	12.87	12.69	12.96
	B	18.4	10.7	10.1	30.7	11.4	20.9	41.4	7.7	8.5	10.9	12.7	17.1	22.9	29.3	44.3	55.8	73.0	101.0	130.6
C164 (640 × 420)	S	1024	729	1024	3125	1296	2401	4096	729	1000	1331	1728	2197	2744	3375	4096	4913	5832	6859	8000
	C	12.60	10.42	10.17	11.35	10.08	10.46	11.12	10.34	10.20	10.76	10.48	10.96	10.66	10.77	11.19	11.18	11.55	11.80	11.30
	B	47.1	28.8	27.9	74.3	27.7	47.3	98.6	19.3	21.1	24.8	29.7	38.7	49.4	69.6	96.2	133.3	179.3	231.1	314.9
HCA (256 × 3129)	S	4096	6561	4096	15625	7776	16807	4096	6561	10000	14641	20736	28561	38416	3375	4096	4913	5832	6859	8000
	C	17.2	15.64	9.79	-	10.47	-	8.54	9.13	9.7	-	-	-	-	8.65	8.52	8.75	9.16	9.07	8.92
	B	121.9	183.6	68.7	-	186.6	-	55.2	115.6	238.9	-	-	-	-	44.3	56.7	70.6	91.9	121.8	156.6
HUEA (256 × 3187)	S	4096	6561	4096	15625	7776	16807	4096	6561	10000	14641	20736	28561	38416	3375	4096	4913	5832	6859	8000
	C	16.02	14.63	8.89	-	9.68	-	7.76	8.46	9.00	-	-	-	-	8.50	7.80	8.69	8.68	8.46	8.27
	B	131.1	189.0	74.4	-	172.3	-	60.3	120.8	245.3	-	-	-	-	49.5	60.4	75.7	95.3	123.3	159.4
101 (60 × 1528)	S	2048	2187	4096	3125	7776	2401	4096	6561	10000	14641	1728	2197	2744	3375	4096	4913	5832	6859	8000
	C	21.99	12.59	17.98	10.25	24.60	7.71	9.33	12.23	16.97	26.49	6.32	7.28	8.28	6.80	7.64	8.54	9.44	10.43	8.25
	B	1021.1	780.5	1728.7	986.5	5167.5	635.6	1426.7	3938.6	7870.7	17973.9	339.7	474.3	658.9	944.0	1498.1	1826.6	2789.8	3543.2	4810.3

\*: w = scene width; h = scene height.

The above experiments were repeated to compare the access time for the different  $k$  values. For each scene, the average time over 100,000 consecutive queries is reported. Results are shown in Table 10, and Figure 6 shows how the access time and the size varied depending on the  $k$  value. As can be observed, access time became smaller and smaller as the value of  $k$  became larger. The plotted data suggested that there was a trade-off between access time and size with respect to the  $k$  value. We considered the optimal  $k$  value to be the one that created a relatively small size with a minimal access time. For example in AG9, when comparing the results between  $k = 6$  and  $k = 15$ , the difference in bits per pixel per band for storage size was not very significant, but the reduction in access time was. Therefore, for this scene,  $k = 15$  was considered an optimal value.



**Figure 6.** A comparison of the storage size (bpppb) and access time ( $\mu$ s) for different  $k$  values of  $k^2$ -raster built from scenes in our datasets. Access time is the average time of 100,000 consecutive queries. For AIRS Granule 9, the best value is marked with a red circle, and the optimal value is marked with a blue square.

**Table 10.** Access time ( $\mu$ s) for a random cell query with different  $k$  values. Each result is the average time over 100,000 consecutive queries. The best results are highlighted in blue.

Scene Data (w × h)*	k = 2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AG9 (90 × 135)	2.45	1.20	0.91	0.82	0.70	0.65	0.61	0.64	0.61	0.57	0.47	0.47	0.48	0.47	0.45	0.42	0.46	0.43	0.43
ACY00 (677 × 512)	15.10	4.95	2.89	2.07	1.60	1.33	1.13	1.01	0.89	0.88	0.79	0.76	0.74	0.73	0.69	0.67	0.68	0.66	0.64
AUY00 (680 × 512)	17.07	5.70	3.03	2.19	1.66	1.40	1.20	1.05	0.94	0.88	0.85	0.83	0.80	0.77	0.72	0.77	0.72	0.70	0.72
C164 (640 × 420)	14.66	5.09	2.84	2.12	1.67	1.48	1.34	1.10	1.08	1.01	0.95	0.93	0.88	0.83	0.81	0.81	0.79	0.74	0.74
HCA (256 × 3129)	0.34	0.26	0.19	-	0.19	-	0.18	0.18	0.16	-	-	-	-	0.17	0.14	0.15	0.15	0.16	0.16
HUEA (256 × 3187)	31.59	10.09	5.24	-	3.11	-	1.87	1.81	1.60	-	-	-	-	1.22	1.02	1.01	1.01	1.00	1.01
I01 (60 × 1528)	6.13	3.35	2.48	2.24	2.13	1.94	1.82	1.76	1.74	1.72	1.62	1.56	1.48	1.55	1.54	1.54	1.41	1.40	1.53

\*: w = scene width; h = scene height.



### 3.2. Heuristic $k^2$ -Raster

In this section, we present the results of the experiments using the heuristic  $k^2$ -raster proposed by Ladra et al. [33] on some of our datasets. Table 11 reports results for two hyperspectral scene datasets: AIRS Granule and AVIRIS Uncalibrated Yellowstone. In the experiments, we found that only when  $k = 2$  would there be enough repeated sets of codewords in the last level of nodes to help us save space. When  $k \geq 3$ , there were no repeated sets of codewords. From the table, it can be seen that there was not much size reduction with  $k_H^2$ -raster in most cases. However, if we built a  $k^2$ -raster using the best or optimal  $k$  value, the size was considerably smaller. Therefore, we can see that  $k_H^2$ -raster structure did not produce a better size.

**Table 11.** Comparison of the structure size (bpppb) built from  $k^2$ -raster and  $k_H^2$ -raster where  $k = 2$ . The sizes for  $k^2$ -raster using the best  $k$  value and the optimal  $k$  value are also shown. The best results are highlighted in blue.

AIRS Granule	$k^2$ -Raster ( $k = 6$ ) (Best)	$k^2$ -Raster ( $k = 15$ ) (Optimal)	$k^2$ -Raster ( $k = 2$ )	$k_H^2$ -Raster ( $k = 2$ )
AG9	9.49	9.53	13.06	13.22
AG16	9.12	9.17	12.72	12.85
AG60	9.81	9.72	13.65	13.86
AG126	9.61	9.72	13.42	13.59
AG129	8.65	8.72	11.98	11.95
AG151	9.53	9.56	13.19	13.35
AG182	9.68	9.71	13.32	13.47
AG193	9.44	9.30	13.29	13.43
AVIRIS Uncalibrated	$k^2$ -Raster ( $k = 9$ ) (Best)	$k^2$ -Raster ( $k = 9$ ) (Optimal)	$k^2$ -Raster ( $k = 2$ )	$k_H^2$ -Raster ( $k = 2$ )
AUY00	11.92	11.92	15.31	15.19
AUY03	11.74	11.74	15.03	14.74
AUY10	9.99	9.99	12.85	11.86
AUY11	11.27	11.27	14.27	14.08
AUY18	12.15	12.15	15.36	15.25

### 3.3. 3D-2D Mapping

As discussed earlier, Cruces et al. [34] proposed a 3D to 2D mapping of raster images using  $k^2$ -tree as an alternative to achieve a better compression ratio. We used the  $k^2$ -tree implementation in sds-lite software to obtain the sizes for one of our datasets (AG9) from  $k = 2$  to  $k = 4$ . Note that similar to  $k^2$ -raster, if the 2D binary matrix cannot be partitioned into square subquadrants of equal size, it needs to be expanded using Equation (1), and the extra elements are set to zero. The results are presented in Table 12. The sizes for a range of bands from 1481 to 1500 of the scene are also given for comparison.

From the results for AG9, we can see that the 3D-2D mapping did not make the size smaller. Instead, it became larger when the  $k$  value increased, and therefore, the method did not produce competitive results.

### 3.4. Comparison of Integer Encoders for $k^2$ -Raster

Experiments were conducted to determine whether other variable-length encoders of integers might serve as a better substitute for DACs, which were the original choice in the  $k^2$ -raster structure initially proposed by Ladra et al. [33]. The performance of DACs was compared to that of other encoders such as Rice, Simple9, PForDelta, Simple16 codes, and gzip. In these experiments, the  $L_{max}$  and  $L_{min}$  arrays were encoded using these codes, and the results are shown in Table 13. For Rice codes, the  $l$  value, as explained in Section 2.4, produced different results depending on the mean of the raster's elements, and only the ones with the best  $l$  value are shown.

**Table 12.** The sizes of AIRS Granule (AG9) produced by 3D to 2D mapping from  $k = 2$  to  $k = 4$ . The individual band sizes ranging from 1481 to 1500 are also shown. Sizes for individual bands are in bits per pixel (bpp), while the ones for all bands are in bits per pixel per band (bpppb).

Band	Original Size	$k^2$ -Tree $k = 2$	$k^2$ -Tree $k = 3$	$k^2$ -Tree $k = 4$
All bands	16	16.53	20.57	26.57
1481	16	17.56	22.00	28.45
1482	16	17.27	21.54	27.84
1483	16	17.19	21.47	27.67
1484	16	17.45	21.81	28.18
1485	16	16.93	21.10	27.29
1486	16	17.09	21.27	27.50
1487	16	16.82	21.06	27.02
1488	16	17.01	21.21	27.34
1489	16	17.23	21.51	27.78
1490	16	16.94	21.10	27.20
1491	16	16.80	20.86	26.96
1492	16	16.56	20.64	26.51
1493	16	16.80	20.91	26.89
1494	16	16.84	20.93	26.98
1495	16	16.69	20.88	26.72
1496	16	16.66	20.75	26.66
1497	16	16.70	20.87	26.73
1498	16	16.61	20.70	26.58
1499	16	16.67	20.73	26.78
1500	16	16.39	20.40	26.18

**Table 13.** A comparison of the storage size (in bpppb) using different integer encoders on  $L_{max}$  and  $L_{min}$  from the  $k^2$ -raster built from our datasets. The combined entropies for  $L_{max}$  and  $L_{min}$  are listed as a reference. The  $l$  value that was used in Rice codes is enclosed in brackets. The best and optimal  $k$  values for DACs are also enclosed in brackets. Except for the entropy, the best rates for each scene's data are highlighted in blue.

Hyperspectral Scene	Entropy ( $L_{max} + L_{min}$ )	Rice ( $l$ Value)	Simple9	PForDelta	Simple16	DACs (Best $k$ )	DACs (Optimal $k$ )	gzip
AG9	8.29	10.10 (7)	10.06	9.88	9.69	9.49 (6)	9.53 (15)	12.45
AG16	7.92	9.88 (7)	9.64	9.55	9.30	9.12 (6)	9.17 (15)	11.96
AG60	8.58	10.31 (7)	10.50	10.19	10.12	9.72 (15)	9.81 (6)	12.79
AG126	8.42	10.34 (7)	10.25	9.98	9.81	9.61 (6)	9.72 (15)	12.55
AG129	7.47	9.66 (7)	9.01	9.01	8.61	8.65 (6)	8.72 (15)	11.21
AG151	8.36	10.39 (7)	9.99	9.79	9.54	9.53 (6)	9.56 (15)	12.39
AG182	8.44	10.58 (7)	10.44	10.09	10.01	9.68 (6)	9.71 (15)	12.71
AG193	8.25	10.26 (7)	10.06	9.93	9.65	9.30 (15)	9.44 (6)	12.33
ACY00	8.81	9.89 (7)	10.37	9.80	10.11	9.61 (6)	9.69 (9)	12.56
ACY03	8.48	9.70 (7)	9.80	9.40	9.57	9.42 (6)	9.50 (9)	11.98
ACY10	6.88	9.18 (7)	7.34	7.43	7.18	7.62 (6)	7.74 (9)	9.32
ACY11	8.12	9.45 (7)	9.32	9.02	9.09	8.81 (6)	9.00 (9)	11.61
ACY18	8.96	10.58 (7)	10.52	9.84	10.28	9.78 (6)	9.88 (9)	12.66
AUY00	11.16	17.59 (7)	14.01	11.93	13.79	11.92 (9)	11.92 (9)	15.13
AUY03	10.83	16.59 (7)	13.54	11.56	13.29	11.74 (9)	11.74 (9)	14.59
AUY10	9.26	12.87 (7)	10.90	9.61	10.54	9.99 (9)	9.99 (9)	12.29
AUY11	10.60	15.16 (7)	13.12	11.24	12.89	11.27 (9)	11.27 (9)	14.47
AUY18	11.38	20.70 (7)	14.19	12.10	14.01	12.15 (9)	12.15 (9)	15.53
C164	9.18	10.33 (7)	11.35	10.44	11.14	10.08 (6)	10.08 (6)	12.85
C165	9.48	10.91 (7)	11.78	10.69	11.57	10.37 (6)	10.37 (6)	13.17
C166	10.02	12.83 (7)	12.99	11.41	12.74	11.05 (6)	11.05 (6)	13.61
C181	9.16	9.96 (7)	10.93	10.53	10.72	9.97 (5)	9.97 (5)	13.37
C182	9.27	10.17 (7)	11.24	10.67	10.99	10.11 (5)	10.11 (5)	13.26
C183	9.60	11.15 (7)	12.33	11.21	12.05	10.65 (5)	10.65 (5)	13.32

Table 13. Cont.

Hyperspectral Scene	Entropy ( $L_{max} + L_{min}$ )	Rice ( $l$ Value)	Simple9	PForDelta	Simple16	DACs (Best $k$ )	DACs (Optimal $k$ )	gzip
HCA	7.59	8.94 (7)	9.79	8.80	9.56	8.52 (16)	8.54 (8)	11.20
HCC	6.75	8.20 (7)	8.28	7.60	7.93	7.62 (8)	7.71 (16)	9.51
HCU	7.87	9.78 (7)	10.30	8.91	10.04	8.85 (16)	8.86 (8)	11.35
HUEA	6.66	7.67 (5)	8.30	7.99	8.00	7.76 (8)	7.80 (16)	9.85
HULM	6.71	7.66 (5)	8.38	8.11	8.10	7.82 (8)	7.88 (16)	10.13
HUMS	6.77	7.90 (5)	8.48	8.14	8.20	7.91 (8)	7.94 (16)	10.12
I01	5.39	6.51 (4)	6.26	6.54	5.94	6.32 (12)	6.80 (15)	7.46
I02	5.46	6.56 (4)	6.27	6.55	5.96	6.38 (12)	6.84 (15)	7.51
I03	5.42	6.51 (4)	6.19	6.48	5.89	6.31 (12)	6.79 (15)	7.39
I04	5.51	6.62 (4)	6.37	6.65	6.04	6.43 (12)	6.90 (15)	7.63

The results showed that, in most cases, DACs still provided the best storage size compared to other encoders for our datasets. They also had the added advantage of direct random access to individual elements of the matrix whilst the other encoders would need to decompress each raster in order to retrieve the element, thus requiring much longer access time. When DACs did not yield the best performance, DACs results were usually only less than 0.1 bpppb worse. In the worst cases, DACs results lagged behind by, at most, 0.4 bpppb.

#### 4. Conclusions

In this research, we examined the possibility of using different integer coding methods for  $k^2$ -raster and concluded that this compact data structure worked best when it was used in tandem with DACs encoding. The other variable-length encoders, though having competitive compression ratios, lacked the ability to provide users with direct access to the data. We also studied a method whereby we could obtain a  $k$  value that gave a competitive storage size and, in most cases, also a suitable access time.

For future work, we are interested in investigating the feasibility of modifying elements in a  $k^2$ -raster structure, facilitating data replacements without having to go through cycles of decompression and compression for the entire compact data structure.

**Author Contributions:** Conceptualization, K.C., D.E.O.T., M.H.-C., I.B., and J.S.-S.; methodology, K.C., D.E.O.T., M.H.-C., I.B., and J.S.-S.; software, K.C.; validation, K.C., I.B., and J.S.-S.; formal analysis, K.C., D.E.O.T., M.H., I.B., and J.S.-S.; investigation, K.C., D.E.O.T., M.H.-C., I.B., and J.S.-S.; resources, K.C., D.E.O.T., M.H.-C., I.B., and J.S.-S.; data curation, K.C., I.B., and J.S.-S.; writing, original draft preparation, K.C., I.B., and J.S.-S.; writing, review and editing, K.C., M.H.-C., I.B., and J.S.-S.; visualization, K.C., I.B., and J.S.-S.; supervision, I.B. and J.S.-S.; project administration, I.B. and J.S.-S.; funding acquisition, M.H.-C., I.B., and J.S.-S. All authors read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund under Grants RTI2018-095287-B-I00 and TIN2015-71126-R (MINECO/FEDER, UE) and BES-2016-078369 (Programa Formación de Personal Investigador), by the Catalan Government under Grant 2017SGR-463, by the postdoctoral fellowship program Beatriu de Pinós, Reference 2018-BP-00008, funded by the Secretary of Universities and Research (Government of Catalonia), and by the Horizon 2020 program of research and innovation of the European Union under the Marie Skłodowska-Curie Grant Agreement #801370.

**Conflicts of Interest:** The authors declare no conflict of interest.

#### References

- Clark, R.N.; Roush, T.L. Reflectance spectroscopy: Quantitative analysis techniques for remote sensing applications. *J. Geophys. Res. Solid Earth* **1984**, *89*, 6329–6340. [[CrossRef](#)]
- Goetz, A.F.; Vane, G.; Solomon, J.E.; Rock, B.N. Imaging spectrometry for earth remote sensing. *Science* **1985**, *228*, 1147–1153. [[CrossRef](#)] [[PubMed](#)]
- Kruse, F.A.; Lefkoff, A.; Boardman, J.; Heidebrecht, K.; Shapiro, A.; Barloon, P.; Goetz, A. The spectral image processing system (SIPS)-interactive visualization and analysis of imaging spectrometer data. *AIP Conf. Proc.* **1993**, *283*, 192–201.

4. Joseph, W. Automated spectral analysis: A geologic example using AVIRIS data, north Grapevine Mountains, Nevada. In Proceedings of the Tenth Thematic Conference on Geologic Remote Sensing, Environmental Research Institute of Michigan, San Antonio, TX, USA, 9–12 May 1994; pp. 1407–1418.
5. Asner, G.P. Biophysical and biochemical sources of variability in canopy reflectance. *Remote Sens. Environ.* **1998**, *64*, 234–253. [[CrossRef](#)]
6. Parente, M.; Kerekes, J.; Heylen, R. A Special Issue on Hyperspectral Imaging [From the Guest Editors]. *IEEE Geosci. Remote Sens. Mag.* **2019**, *7*, 6–7. [[CrossRef](#)]
7. Ientilucci, E.J.; Adler-Golden, S. Atmospheric Compensation of Hyperspectral Data: An Overview and Review of In-Scene and Physics-Based Approaches. *IEEE Geosci. Remote Sens. Mag.* **2019**, *7*, 31–50. [[CrossRef](#)]
8. Khan, M.J.; Khan, H.S.; Yousaf, A.; Khurshid, K.; Abbas, A. Modern trends in hyperspectral image analysis: A review. *IEEE Access* **2018**, *6*, 14118–14129. [[CrossRef](#)]
9. Theiler, J.; Ziemann, A.; Matteoli, S.; Diani, M. Spectral Variability of Remotely Sensed Target Materials: Causes, Models, and Strategies for Mitigation and Robust Exploitation. *IEEE Geosci. Remote Sens. Mag.* **2019**, *7*, 8–30. [[CrossRef](#)]
10. Sun, W.; Du, Q. Hyperspectral band selection: A review. *IEEE Geosci. Remote Sens. Mag.* **2019**, *7*, 118–139. [[CrossRef](#)]
11. Scafutto, R.D.M.; de Souza Filho, C.R.; de Oliveira, W.J. Hyperspectral remote sensing detection of petroleum hydrocarbons in mixtures with mineral substrates: Implications for onshore exploration and monitoring. *ISPRS J. Photogramm. Remote Sens.* **2017**, *128*, 146–157. [[CrossRef](#)]
12. Bishop, C.A.; Liu, J.G.; Mason, P.J. Hyperspectral remote sensing for mineral exploration in Pulang, Yunnan Province, China. *Int. J. Remote Sens.* **2011**, *32*, 2409–2426. [[CrossRef](#)]
13. Le Marshall, J.; Jung, J.; Zapotocny, T.; Derber, J.; Treadon, R.; Lord, S.; Goldberg, M.; Wolf, W. The application of AIRS radiances in numerical weather prediction. *Aust. Meteorol. Mag.* **2006**, *55*, 213–217.
14. Robichaud, P.R.; Lewis, S.A.; Laes, D.Y.; Hudak, A.T.; Kokaly, R.F.; Zamudio, J.A. Postfire soil burn severity mapping with hyperspectral image unmixing. *Remote Sens. Environ.* **2007**, *108*, 467–480. [[CrossRef](#)]
15. Navarro, G. *Compact Data Structures: A Practical Approach*; Cambridge University Press: Cambridge, UK, 2016.
16. Jacobson, G. Space-efficient static trees and graphs. In Proceedings of the 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC, USA, 30 October–1 November 1989; pp. 549–554.
17. Consultative Committee for Space Data Systems (CCSDS). *Image Data Compression CCSDS 123.0-B-1*; Blue Book; CCSDS: Washington, DC, USA, 2012.
18. Jolliffe, I.T. *Principal Component Analysis*; Springer: Berlin, Germany, 2002; p. 487.
19. Taubman, D.S.; Marcellin, M.W. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*; Kluwer Academic Publishers: Boston, MA, USA, 2001.
20. Chow, K.; Tzamaras, D.E.O.; Blanes, I.; Serra-Sagristà, J. Using Predictive and Differential Methods with K2-Raster Compact Data Structure for Hyperspectral Image Lossless Compression. *Remote Sens.* **2019**, *11*, 2461. [[CrossRef](#)]
21. Elias, P. Efficient storage and retrieval by content and address of static files. *J. ACM (JACM)* **1974**, *21*, 246–260. [[CrossRef](#)]
22. Ottaviano, G.; Venturini, R. Partitioned Elias-Fano indexes. In Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, Gold Coast, Australia, 6–11 July 2014; pp. 273–282.
23. Pibiri, G.E. Dynamic Elias-Fano Encoding. Master’s Thesis, University of Pisa, Pisa, Italy, 2014.
24. Sugiura, R.; Kamamoto, Y.; Harada, N.; Moriya, T. Optimal Golomb-Rice Code Extension for Lossless Coding of Low-Entropy Exponentially Distributed Sources. *IEEE Trans. Inf. Theory* **2018**, *64*, 3153–3161. [[CrossRef](#)]
25. Rice, R.; Plaunt, J. Adaptive variable-length coding for efficient compression of spacecraft television data. *IEEE Trans. Commun. Technol.* **1971**, *19*, 889–897. [[CrossRef](#)]
26. Rojals, J.S.; Karczewicz, M.; Joshi, R.L. Rice Parameter Update for Coefficient Level Coding in Video Coding Process. U.S. Patent 9,936,200, 3 April 2018.
27. Zukowski, M.; Heman, S.; Nes, N.; Boncz, P. Super-scalar RAM-CPU cache compression. In Proceedings of the 22nd International Conference on Data Engineering (ICDE’06), Atlanta, GA, USA, 3–7 April 2006; pp. 59–59.
28. Silva-Coira, F. Compact Data Structures for Large and Complex Datasets. Ph.D. Thesis, Universidade da Coruña, A Coruña, Spain, 2017.

29. Al Hasib, A.; Cebrian, J.M.; Natvig, L. V-PFORDelta: Data compression for energy efficient computation of time series. In Proceedings of the 2015 IEEE 22nd International Conference on High Performance Computing (HiPC), Bengaluru, India, 16–19 December 2015; pp. 416–425.
30. Brisaboa, N.R.; Ladra, S.; Navarro, G. DACs: Bringing direct access to variable-length codes. *Inf. Process. Manag.* **2013**, *49*, 392–404. [[CrossRef](#)]
31. Brisaboa, N.R.; Ladra, S.; Navarro, G. Directly addressable variable-length codes. In Proceedings of the International Symposium on String Processing and Information Retrieval, Saariselkä, Finland, 25–27 August 2009; pp. 122–130.
32. Baruch, G.; Klein, S.T.; Shapira, D. A space efficient direct access data structure. *J. Discret. Algorithms* **2017**, *43*, 26–37. [[CrossRef](#)]
33. Ladra, S.; Paramá, J.R.; Silva-Coira, F. Scalable and queryable compressed storage structure for raster data. *Inf. Syst.* **2017**, *72*, 179–204. [[CrossRef](#)]
34. Cruces, N.; Seco, D.; Gutiérrez, G. A compact representation of raster time series. In Proceedings of the Data Compression Conference (DCC), Snowbird, UT, USA, 26–29 March 2019; pp. 103–111.
35. Brisaboa, N.R.; Ladra, S.; Navarro, G. k 2-trees for compact web graph representation. In Proceedings of the International Symposium on String Processing and Information Retrieval, Saariselkä, Finland, 25–27 August 2009; pp. 18–30.
36. Anh, V.N.; Moffat, A. Inverted index compression using word-aligned binary codes. *Inf. Retr.* **2005**, *8*, 151–166. [[CrossRef](#)]
37. Zhang, J.; Long, X.; Suel, T. Performance of compressed inverted list caching in search engines. In Proceedings of the 17th International Conference on World Wide Web, Beijing, China, 21–25 April 2008; pp. 387–396.
38. Gog, S.; Beller, T.; Moffat, A.; Petri, M. From theory to practice: Plug and play with succinct data structures. In Proceedings of the International Symposium on Experimental Algorithms, Copenhagen, Denmark, 29 June–1 July 2014; pp. 326–337.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).

## Chapter 4

# Performance improvement on $k^2$ -raster compact data structure for hyperspectral scenes



# Performance Improvement on $k^2$ -Raster Compact Data Structure for Hyperspectral Scenes

Kevin Chow<sup>1</sup>, Dion Eustathios Olivier Tzarmarias, Miguel Hernández-Cabronero<sup>2</sup>,  
Ian Blanes<sup>1</sup>, *Senior Member, IEEE*, and Joan Serra-Sagrà<sup>1</sup>, *Senior Member, IEEE*

**Abstract**—This letter proposes methods to improve data size and access time for  $k^2$ -raster, a losslessly compressed data structure that provides efficient storage and real-time processing. Hyperspectral scenes from real missions are used as our testing data. In previous studies, with  $k^2$ -raster, the size of the hyperspectral data was reduced by up to 52% compared with the uncompressed data. In this letter, we continue to explore novel ways of further reducing the data size and access time. First, we examine the possibility of using the raster matrix of hyperspectral data without any padding (unpadded matrix) while still being able to compress the structure and access the data. Second, we examine some integer encoders, more specifically the Simple family. We discuss their ability to provide random element access and compare them with directly addressable codes (DACs), the integer encoder used in the original description for  $k^2$ -raster. Experiments show that the use of unpadded matrices has improved the storage size up to 6% while the use of a different integer encoder reduces the storage size up to 6% and element access time up to 20%.

**Index Terms**—Directly addressable codes (DACs), image compression, lossless hyperspectral imaging, PForDelta, remote sensing, Simple-9, Simple-16.

## I. INTRODUCTION

**H**YPERSPECTRAL scenes are data taken from the air by sensors, such as airborne visible/infrared imaging spectrometer (AVIRIS), or from space by satellite instruments such as Hyperion and infrared atmospheric sounding interferometer (IASI). These scenes are made up of multiple bands from across the electromagnetic spectrum, and data extracted from certain bands have many practical applications, such as oil field exploration and mineral exploration. Due to their relatively large sizes, hyperspectral scenes are usually compressed to increase transmission throughput and reduce data volumes.

Compact data structures can store data efficiently and provide real-time data compression and access to the

original data [1]. They can be loaded into main memory, and operations to access data are often carried out by means of the rank and select functions [2]. Compact data structures provide reduced space usage and query time. There is no need to decompress a large portion of the structure to access and query individual data as it is the case with data compressed by classical compression algorithms such as gzip and specialized algorithms such as CCSDS 123.0-B-2 [3].

In this work, we reduce the hyperspectral data size using  $k^2$ -raster, a compact data structure, to produce lossless compression. In our previous letter [4], we presented a predictive method and a differential method that made use of spatial and spectral correlations in hyperspectral data with favorable results. Nevertheless, due to the nature of these methods, only random access to individual cells can be done, whereas other operations such as query on a region cannot be performed. In this letter, we focus on investigating whether unpadded matrices and variable-length integer encoders other than directly addressable codes (DACs) [5] can provide competitive compression ratios as well while improving random and query access time. In our case, we need to store non-negative small integers in the  $k^2$ -raster tree structure, which is built in such a way that the nodes are not connected by pointers but can still be reached with the use of a compact data structure's linear rank function. Fig. 1 depicts a global picture of the interrelations between the elements discussed above. The compact data structures that we have been working on are still at the research stage when applied to hyperspectral scenes, but with the encouraging results that we have obtained so far, we can extrapolate their practical use in applications of remote sensing and geographic information systems [6]–[10].

The letter is organized as follows. Section II provides background information on  $k^2$ -raster built from a padded matrix and the various integer encoders. Section III describes the proposed method of using an unpadded matrix to build the structure and introduces the different variable-length integer encoders. Section IV presents some experimental results. Section V sums up the key points of our discussion.

## II. BACKGROUND

Ladra *et al.* [11] proposed  $k^2$ -raster, a tree structure specifically designed for raster data including images. It is built from a matrix of width  $w$  and height  $h$ , and an integer  $k \geq 2$ . If the matrix can be partitioned into  $k^2$  square quadrants of equal size, it can be used directly. Otherwise, it is necessary to enlarge the matrix to size  $s \times s$ , where  $s = k^{\lceil \log_k \max(w, h) \rceil}$ , and the number of subdivisions is  $\log_k(s)$ . The padding elements are equal to zero. This extended (padded) matrix is then recursively partitioned into  $k^2$  square submatrices of identical size, hereafter referred to as quadrants. This process is repeated

Manuscript received January 15, 2021; revised April 29, 2021; accepted May 17, 2021. This work was supported in part by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund (Programa Formación de Personal Investigador) under Grant RTI2018-095287-B-I00 and Grant BES-2016-078369, in part by the Catalan Government under Grant 2017SGR-463, in part by the Postdoctoral Fellowship Program Beatriu de Pinós funded by the Secretary of Universities and Research (Government of Catalonia) under Grant 2018-BP-00008, and in part by the Horizon 2020 Program of Research and Innovation of the European Union under the Marie Skłodowska-Curie Grant Agreement 801370. (Corresponding author: Kevin Chow.)

The authors are with the Department of Information and Communications Engineering, Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain (e-mail: kevin.chow@uab.cat).

Color versions of one or more figures in this letter are available at <https://doi.org/10.1109/LGRS.2021.3084065>.

Digital Object Identifier 10.1109/LGRS.2021.3084065

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see <https://creativecommons.org/licenses/by/4.0/>

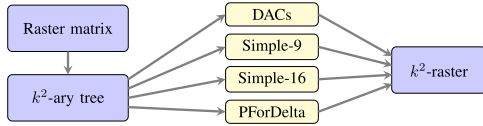


Fig. 1. Construction of  $k^2$ -raster. A  $k^2$ -ary tree is first built from a raster matrix. Compression and random access are achieved when tree node data are encoded by an integer encoder, such as DACs, Simple-9, Simple-16, or PForDelta, resulting in a  $k^2$ -raster structure.

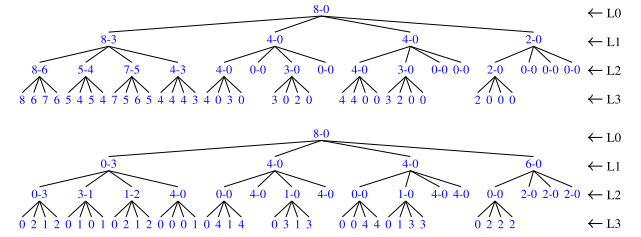
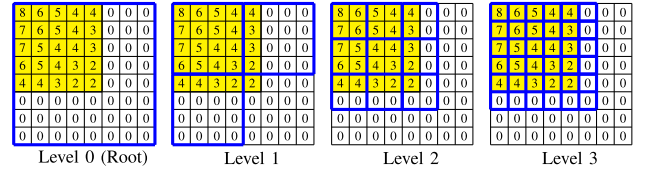
until all cells in a quadrant have the same value, or until the submatrix has size  $1 \times 1$  and cannot be further subdivided. This partitioning produces the nodes for a  $k^2$ -ary tree topology where the data in the nodes are stored in the following data structures.

- 1) At each tree level  $\ell$ , the maximum and minimum values of each quadrant are computed. These are then compared with the corresponding maximum and minimum values in the parent node, and the differences are stored in the  $V_{\max_\ell}$  and  $V_{\min_\ell}$  arrays of each level. Saving the differences instead of the original values results in smaller values for each node, which in turn allows a better compression with an integer encoder. Next, with the exception of the root node at the top level, the  $V_{\max_\ell}$  and  $V_{\min_\ell}$  arrays at all the levels are concatenated to form  $L_{\max}$  and  $L_{\min}$ , respectively. Both arrays are then compressed by an integer encoder.
- 2) The root's maximum ( $rMax$ ) and minimum ( $rMin$ ) values are stored as uncompressed integers.
- 3) A bitmap array  $T$  is generated from all the nodes except the ones at the root and at the last level, each node indicating whether it has child nodes or not. This bitmap serves to locate the tree nodes when cell queries are performed by means of a rank function [2].

In Fig. 2, an example of a  $5 \times 5$  matrix is shown to illustrate this process. The elements which fully describe the resulting  $k^2$ -raster structure are shown at the bottom of Fig. 2. Please refer to [12] for a more comprehensive description of  $k^2$ -raster.

DACs were proposed by Brisaboa *et al.* [5]. Consider a sequence of integers  $x$ . Each integer  $x_i$ , which is represented by  $\lfloor \log_2 x_i \rfloor + 1$  bits, is broken into chunks of bits of size  $C_S$ . Each chunk is stored in a block of size  $C_S + 1$  with the additional (highest) bit used as a control bit (0 for most significant bits, 1 otherwise). More information on DACs and software implementation can be found in the paper by Brisaboa *et al.* [5].

Simple-9 word-aligned encoding [13] is another approach to compression. Each 32-bit word is split into two parts: a 28-bit part where a variable number of integers are encoded and a 4-bit part which is a selector with a value ranging from 0 to 8. For example, selector 0 signals that the first 28 integers in the data have a value of 0 or 1. Selector 1 signals that it can pack 14 integers into the segment with a maximum bit length of 2 bits for each. Beginning with selector 0, each selector is tested until the most efficient one is found. Simple-16 [14] is a variant of Simple-9 and uses all the 16 combinations in the selector bits. Their values range from 0 to 15. PForDelta [15] is also similar to both Simple-9 and Simple-16 but encodes a fixed group of 32, 64, 128, or 256 integers in a varying number of bytes. A predetermined percentage of those numbers that are larger than the others are encoded separately after the smaller numbers or in another location.



Element	Bin/Dec	Tree Level	Node Data
$L_{\max}$	Decimal	Level 1	0446
		Level 2	0314 0414 0144 0222
		Level 3	0212 0101 0212 0001 0414 0313 0044 0133 0222
$L_{\min}$	Decimal	Level 1	3000
		Level 2	3120 00 00 0
$rMax$	Decimal		8
$rMin$	Decimal		0
$T$ Bitmap	Binary		1111 1111 1010 1100 1000

Fig. 2. Top: A  $5 \times 5$  matrix example showing recursive partitioning. Middle: The upper tree is a  $k^2$ -raster ( $k = 2$ ) tree constructed from the matrix and the lower tree takes into account the differences between the parent and child nodes. Bottom: A table showing the elements of the  $k^2$ -raster with padding.

### III. PROPOSAL

#### A. Proposed Building $k^2$ -Raster Without Padding

As mentioned in Section II, given a  $k$  value, a matrix with a size that is not a power of  $k^2$  needs to be extended according to the equation for computing  $s$  in that section. The values of the new cells are set to zero. This is necessary because the search from the root down to its leaves requires the knowledge of its child node location using the rank function, which is a function of the number of child nodes each parent node has. Adding new cells, however, also means that the nodes that are outside the matrix have to be saved, and this leads to a larger structure.

To illustrate the above point, we construct a tree based on the elements within the bounds of the original matrix where  $k = 2$ . This matrix together with its corresponding  $k^2$ -raster tree are shown in Fig. 3. This is done without padding and is therefore not a full quaternary tree (a full quaternary tree is one where each node has either 0 or 4 child nodes). In this case, the parent does not know how many children it has, so it is not possible to use the rank function to get to the correct child nodes without including them in the  $T$  bitmap. On the other hand, with padding, as explained earlier in Section II, it is a full quaternary tree. Fig. 2 shows the padded matrix and the tree that is built from it. Comparing it with Fig. 3, we can see that the  $k^2$ -raster tree from the unpadded matrix can save fewer elements.

To build a non-full quaternary tree, we modify the original function for building the tree and prune the values that are located outside the bounds of the original matrix. This reduces the number of  $L_{\max}$  and  $L_{\min}$  values that need to be saved. However, when forming the  $T$  bitmap, the full-quaternary tree is still used to ensure that the parent nodes can reach their child nodes, with the corresponding bit value of the node that



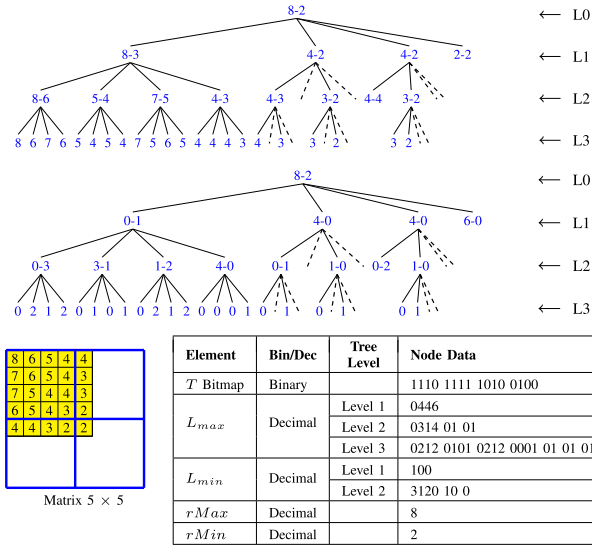


Fig. 3. Top: A non-full quaternary tree constructed from a  $5 \times 5$  matrix without padding. The second and third parent nodes at Level 1 have only two child nodes each. All nodes that are outside the bounds are connected by dashed lines and will not be saved, and the corresponding value in  $T$  bitmap is set to zero. Bottom: The  $5 \times 5$  matrix and a table showing the elements of the  $k^2$ -raster without padding.

is outside the bounds being set to zero. This facilitates the search path through the full quaternary tree when cell queries are performed using the rank function. Note that the  $T$  bitmap does not have to store the location information of the nodes of the last level because we can compute the location of the values from the original matrix size and the  $s$  value in the expanded matrix.

The algorithms for building  $k^2$ -raster with and without padding are listed in Algorithm 1. What this build function does is it excludes elements that are outside the bounds of the original matrix and save only the actual data. This helps reduce the size of the structure. To compute the theoretical storage reduction, we can count how many symbols 0 we are sparing in the encoding with the unpadded matrix:  $\text{spared0}$ . Since the actual reduction in storage depends on the entropy encoding, we could estimate the saving to be  $\text{spared0} \times \log_2(\text{Probability}(\text{spared0}))$ . Another way to estimate space saving is to calculate the size of the original raster matrix  $\text{originalDim}$  compared with that of the expanded matrix  $\text{expandedDim}$ . The maximum saving can be estimated to be  $((\text{expandedDim} - \text{originalDim}) / \text{expandedDim}) \times 100\%$  but the saving is, in general, less, due to factors such as the  $k$  value, tree height, and  $k^2$ -raster saving such as nodes at a higher level that become leaves. For example, in Figs. 2 and 3, the estimated maximum saving is  $((64 - 25) / 64) \times 100\% = 61\%$  and the  $L_{max}$  nodes' saving is actually  $((56 - 34) / 56) \times 100\% = 39\%$ . Hence, there is a relationship between the image dimensions and the storage saving.

### B. Random Access for Integer Encoders

In saving the  $L_{max}$  and  $L_{min}$  arrays, the authors of  $k^2$ -raster made use of DACs as an integer encoder for random access to its elements. In this research, we have investigated other word-aligned integer encoders from the Simple family: Simple-9 and Simple-16, and the PForDelta variant, which also allow random access due to their structure.

**Algorithm 1:** Algorithm for Constructing  $T$ ,  $Vmax$ ,  $Vmin$  for a Padded Matrix and an Unpadded Matrix. With the “+” Lines Removed, the Pseudocode Is for the Function **Build**( $n, nx, ny, l, r, c, o_l$ ) for Building From an Unpadded Matrix  $M$ . With the “-” Lines Removed From the Code and the “+” Lines Re-Added, It Becomes One for the Function **Build**( $n, l, r, c$ ) From a Padded Matrix  $M$

```

maxval ← 0
minval ← ∞
-outcount ← 0
for i ← 0...k - 1 do
  for j ← 0...k - 1 do
    if n = k then /* last level */
      if (r + i) < nx and (c + j) < ny then
        | o_l[pmax_l] ← 0
      else
        | o_l[pmax_l] ← 1
        - outcount ← outcount + o_l[pmax_l]
        - maxval ← max(maxval, M_{r+i,c+j})
        - minval ← min(minval, M_{r+i,c+j})
        - Vmax_l[pmax_l] ← M_{r+i,c+j}
        - pmax_l ← pmax_l + 1
    else /* in-between level */
      if (r + i · (n/k)) < nx and (c + j · (n/k)) < ny
      then
        | o_l[pmax_l] ← 0
      else
        | o_l[pmax_l] ← 1
        - outcount ← outcount + o_l[pmax_l]
        - (childmax, childmin, childoutcount) ←
        - Build(n/k, nx, ny, l + 1, r + i · (n/k), c + j ·
        - (n/k), o_l)
        + (childmax, childmin) ←
        + Build(n/k, l + 1, r + i · (n/k), c + j · (n/k))
        - Vmax_l[pmax_l] ← childmax
        - Vmin_l[pmin_l] ← childmin
      if
        - childoutcount = k2 or childmax = childmin
        then
        + if childmax = childmin then
        + | T_l[pmax_l] ← 0
        else
        + | T_l[pmax_l] ← 1
        + pmax_l ← pmax_l + 1
        + pmin_l ← pmin_l + 1
        + maxval ← max(maxval, childmax)
        + minval ← min(minval, childmin)
    + if maxval = minval then
    + | pmax_l ← pmax_l - k2
    + | pmin_l ← pmin_l - k2
    + return (maxval, minval)
  - return (maxval, minval, outcount)

```

The use of partial sums and sampling described in [1, §3.3 and §4.2] can be used in the Simple family of codes to expedite the search process in the compressed array. We incorporate such strategies by sampling these arrays at

TABLE I

TESTING SCENE DATA USED AND A COMPARISON OF  $k^2$ -RASTER STORAGE SIZE WITH PADDED MATRIX AND UNPADDED MATRIX. SIZE REDUCTIONS (%) FROM PADDED MATRIX TO UNPADDED MATRIX ARE ENCLOSED IN PARENTHESES. IN THE DIMENSIONS COLUMN  $x$  IS THE IMAGE WIDTH,  $y$  THE IMAGE HEIGHT, AND  $z$  THE NUMBER OF SPECTRAL BANDS. THE BEST RESULTS ARE HIGHLIGHTED.

Sensor	Name	Acro- nym	Original Dimensions (x × y × z)	Gzip Rate (b/pppb)	Best k Value	Padded Matrix (b/pppb)					Unpadded Matrix (b/pppb)															
						DACs	S9*	S16*	PFD* 32	PFD* 64	PFD* 128	PFD* 256	DACs	S9*	S16*	PFD* 32	PFD* 64	PFD* 128	PFD* 256							
AIRS	9 <sup>†</sup>	AG9	90×135×1501	10.16	6	<b>9.49</b>	10.06	9.69	9.88	9.59	9.59	9.82	8.94	(5.8)	9.44	(6.1)	9.08	(6.3)	9.28	(6.1)	8.94	(6.8)	<b>8.90</b>	(7.2)	9.07	(7.6)
	16 <sup>†</sup>	AG16	90×135×1501	9.82	6	<b>9.12</b>	9.64	9.30	9.55	9.20	9.15	9.44	8.60	(5.7)	9.02	(6.4)	8.68	(6.7)	8.95	(6.2)	8.55	(7.0)	<b>8.45</b>	(7.7)	8.53	(9.6)
	60 <sup>†</sup>	AG60	90×135×1501	10.53	6	9.81	10.50	10.12	10.19	9.82	<b>9.75</b>	10.00	9.28	(5.4)	9.89	(5.7)	9.51	(5.9)	9.65	(5.3)	9.22	(6.1)	<b>9.10</b>	(6.7)	9.19	(8.1)
	126 <sup>†</sup>	AG126	90×135×1501	10.33	6	9.61	10.25	9.81	9.98	9.61	<b>9.53</b>	9.84	9.07	(5.6)	9.65	(5.9)	9.21	(6.2)	9.41	(5.7)	9.00	(6.4)	<b>8.85</b>	(7.1)	9.01	(8.4)
	129 <sup>†</sup>	AG129	90×135×1501	9.50	6	8.65	9.01	<b>8.61</b>	9.01	8.69	8.67	8.95	8.10	(6.3)	8.39	(7.0)	7.98	(7.3)	8.39	(6.9)	8.01	(6.8)	<b>7.92</b>	(8.7)	7.98	(10.9)
	151 <sup>†</sup>	AG151	90×135×1501	10.31	6	9.53	9.99	9.54	9.79	9.43	<b>9.41</b>	9.79	8.97	(5.8)	9.39	(6.0)	8.94	(6.3)	9.16	(6.4)	8.78	(6.9)	<b>8.71</b>	(7.4)	8.84	(9.7)
	182 <sup>†</sup>	AG182	90×135×1501	10.64	6	<b>9.68</b>	10.44	10.01	10.09	9.79	9.77	10.08	9.14	(5.6)	9.86	(5.6)	9.42	(5.9)	9.53	(5.6)	9.20	(6.1)	<b>9.11</b>	(6.8)	9.21	(8.6)
	193 <sup>†</sup>	AG193	90×135×1501	10.15	6	<b>9.44</b>	10.06	9.65	9.93	9.56	9.46	9.74	8.90	(5.7)	9.45	(6.1)	9.03	(6.4)	9.33	(6.0)	8.89	(7.0)	<b>8.72</b>	(7.8)	8.80	(9.7)
	Average			10.18		<b>9.42</b>	9.99	9.59	9.80	9.46	<b>9.42</b>	9.71	8.88	(5.7)	9.39	(6.1)	8.98	(6.4)	9.21	(6.0)	8.82	(6.7)	<b>8.72</b>	(7.4)	8.83	(9.1)
AVIRIS	Yellowstone sc. 00 <sup>†</sup>	AC00	677×512×224	10.12	6	9.61	10.37	10.11	9.80	9.41	<b>9.35</b>	9.40	9.47	(1.5)	10.20	(1.6)	9.95	(1.6)	9.67	(1.3)	9.28	(1.4)	<b>9.21</b>	(1.5)	9.26	(1.5)
	Yellowstone sc. 03 <sup>†</sup>	AC03	677×512×224	9.59	6	9.42	9.80	9.57	9.40	9.03	<b>8.99</b>	9.07	9.29	(1.4)	9.65	(1.6)	9.42	(1.6)	9.28	(1.3)	8.92	(1.2)	<b>8.86</b>	(1.4)	8.94	(1.4)
	Yellowstone sc. 10 <sup>†</sup>	AC10	677×512×224	7.41	6	7.62	7.34	7.18	7.44	7.06	<b>7.04</b>	7.14	7.49	(1.8)	7.20	(2.0)	7.04	(2.0)	7.31	(1.7)	6.94	(1.6)	<b>6.90</b>	(1.9)	7.02	(1.7)
	Yellowstone sc. 11 <sup>†</sup>	AC11	677×512×224	9.04	6	8.81	9.32	9.09	9.02	8.65	<b>8.62</b>	8.72	8.67	(1.7)	9.17	(1.6)	8.94	(1.7)	8.90	(1.3)	8.52	(1.5)	<b>8.49</b>	(1.5)	8.59	(2.1)
	Yellowstone sc. 18 <sup>†</sup>	AC18	677×512×224	10.00	6	9.78	10.52	10.28	9.84	9.47	<b>9.42</b>	9.50	9.65	(1.3)	10.37	(1.4)	10.14	(1.4)	9.73	(1.1)	9.35	(1.2)	<b>9.30</b>	(1.3)	9.38	(1.3)
	Average			9.23		9.05	9.47	9.25	9.10	8.72	<b>8.68</b>	8.76	8.91	(1.5)	9.32	(1.6)	9.10	(1.6)	8.98	(1.3)	8.60	(1.4)	<b>8.55</b>	(1.5)	8.63	(1.5)
	Yellowstone sc. 00 <sup>†</sup>	AU00	680×512×224	12.39	9	11.92	14.01	13.79	11.93	11.54	<b>11.44</b>	11.55	11.75	(1.4)	13.83	(1.2)	13.62	(1.2)	11.75	(1.6)	11.33	(1.8)	<b>11.22</b>	(1.9)	11.30	(2.2)
	Yellowstone sc. 03 <sup>†</sup>	AU03	680×512×224	11.98	9	11.74	13.54	13.29	11.56	11.15	<b>11.08</b>	11.22	11.58	(1.4)	13.37	(1.3)	13.12	(1.3)	11.37	(1.6)	10.95	(1.8)	<b>10.87</b>	(2.0)	10.97	(2.2)
	Yellowstone sc. 10 <sup>†</sup>	AU10	680×512×224	10.17	9	9.99	10.90	10.54	9.61	9.20	<b>9.10</b>	9.26	9.82	(1.7)	10.72	(1.6)	10.36	(1.7)	9.42	(2.0)	8.98	(2.4)	<b>8.87</b>	(2.6)	8.99	(2.9)
Yellowstone sc. 11 <sup>†</sup>	AU11	680×512×224	11.49	9	11.27	13.12	12.89	11.24	10.85	<b>10.77</b>	10.94	11.08	(1.7)	12.94	(1.4)	12.71	(1.4)	11.05	(1.7)	10.64	(1.9)	<b>10.56</b>	(2.0)	10.69	(2.2)	
Yellowstone sc. 18 <sup>†</sup>	AU18	680×512×224	12.29	9	12.15	14.19	14.01	12.10	11.70	<b>11.63</b>	11.75	11.99	(1.3)	14.01	(1.2)	13.84	(1.3)	11.91	(1.6)	11.50	(1.8)	<b>11.41</b>	(1.9)	11.50	(2.1)	
Average			11.66		11.42	13.15	12.91	11.29	10.89	<b>10.80</b>	10.94	11.24	(1.5)	12.98	(1.3)	12.73	(1.4)	11.10	(1.7)	10.68	(1.9)	<b>10.58</b>	(2.0)	10.69	(2.3)	
CRISM	frf000065e6_07_sc164 <sup>†</sup>	CR1	640×420×545	10.98	6	<b>10.08</b>	11.35	11.14	10.44	10.22	10.37	10.54	<b>10.00</b>	(0.8)	11.09	(2.2)	10.89	(2.3)	10.36	(0.7)	10.14	(0.8)	10.29	(0.8)	10.45	(0.9)
	frf00008849_07_sc165 <sup>†</sup>	CR2	640×420×545	11.03	6	<b>10.37</b>	11.78	11.57	10.69	10.49	10.65	10.84	<b>10.29</b>	(0.8)	11.69	(0.8)	11.48	(0.8)	10.62	(0.7)	10.42	(0.7)	10.57	(0.7)	10.76	(0.7)
	frf0001077d_07_sc166 <sup>†</sup>	CR3	640×480×545	11.20	6	<b>11.05</b>	12.99	12.74	11.41	11.22	11.35	11.49	<b>10.97</b>	(0.7)	12.89	(0.8)	12.64	(0.8)	11.35	(0.5)	11.15	(0.7)	11.27	(0.7)	11.39	(0.8)
	hrf00004f38_07_sc181 <sup>†</sup>	CR4	320×420×545	10.77	5	<b>9.97</b>	10.93	10.72	10.53	10.30	10.37	10.34	<b>9.90</b>	(0.7)	10.88	(0.4)	10.67	(0.5)	10.48	(0.4)	10.24	(0.6)	10.31	(0.6)	10.28	(0.6)
	hrf0000648f_07_sc182 <sup>†</sup>	CR5	320×420×545	10.90	5	<b>10.11</b>	11.24	10.99	10.67	10.47	10.53	10.50	<b>10.06</b>	(0.5)	11.21	(0.2)	10.97	(0.3)	10.64	(0.3)	10.43	(0.4)	10.49	(0.4)	10.46	(0.4)
	hrf0000ba9c_07_sc183 <sup>†</sup>	CR6	320×480×545	10.87	5	<b>10.65</b>	12.33	12.05	11.21	11.01	11.04	10.99	<b>10.57</b>	(0.7)	12.27	(0.4)	11.99	(0.5)	11.15	(0.5)	10.95	(0.5)	10.97	(0.7)	10.92	(0.7)
	Average			10.96		<b>10.37</b>	11.77	11.54	10.82	10.62	10.72	10.78	<b>10.30</b>	(0.7)	11.67	(0.8)	11.44	(0.8)	10.77	(0.5)	10.56	(0.6)	10.65	(0.7)	10.71	(0.7)
Hyperion	Agricultural <sup>†</sup>	HC1	256×3129×242	8.84	8	8.54	9.79	9.56	8.80	8.42	<b>8.35</b>	8.37	8.52	(0.3)	9.77	(0.2)	9.54	(0.2)	8.86	(-0.7)	8.49	(-0.9)	<b>8.36</b>	(-0.2)	8.36	(0.1)
	Coral Reef <sup>†</sup>	HC2	256×3127×242	7.45	8	7.62	8.28	7.93	7.60	7.18	<b>7.08</b>	7.10	7.62	(0.1)	8.28	(0.1)	7.93	(0.1)	7.67	(-0.8)	7.29	(-1.5)	<b>7.15</b>	(-1.1)	<b>7.15</b>	(-0.7)
	Urban <sup>†</sup>	HC3	256×2905×242	8.85	8	8.86	10.30	10.04	8.91	8.51	<b>8.46</b>	8.50	8.83	(0.3)	10.28	(0.2)	10.02	(0.2)	8.93	(-0.2)	8.51	(-0.0)	<b>8.44</b>	(0.3)	8.48	(0.3)
	Average			8.38		8.34	9.46	9.18	8.44	8.04	<b>7.96</b>	7.99	8.32	(0.2)	9.44	(0.2)	9.16	(0.2)	8.49	(-0.6)	8.10	(-0.7)	<b>7.99</b>	(-0.3)	8.00	(-0.1)
	Erta Ale <sup>†</sup>	HU1	256×3187×242	8.69	8	7.76	8.30	8.00	7.99	7.47	<b>7.32</b>	7.33	7.73	(0.5)	8.27	(0.4)	7.97	(0.4)	8.05	(-0.7)	7.56	(-1.2)	7.37	(-0.6)	<b>7.32</b>	(0.2)
Lake Montana <sup>†</sup>	HU2	256×3176×242	8.69	8	7.82	8.38	8.10	8.11	7.60	<b>7.46</b>	7.47	7.80	(0.2)	8.37	(0.1)	8.08	(0.1)	8.19	(-1.0)	7.71	(-1.3)	7.51	(-0.7)	<b>7.50</b>	(-0.3)	
Mt. St. Helena <sup>†</sup>	HU3	256×3242×242	8.26	8	7.91	8.48	8.20	8.14	7.63	<b>7.50</b>	7.53	7.87	(0.5)	8.44	(0.4)	8.17	(0.5)	8.20	(-0.7)	7.74	(-1.4)	7.55	(-0.6)	<b>7.52</b>	(0.1)	
Average			8.55		7.83	8.38	8.10	8.08	7.57	<b>7.43</b>	7.45	7.80	(0.4)	8.36	(0.3)	8.07	(0.3)	8.15	(-0.8)	7.67	(-1.3)	7.47	(-0.6)	<b>7.44</b>	(0.0)	
IASI	Level 0 1 <sup>†</sup>	IAS11	60×1528×8359	5.90	12	6.32	6.26	<b>5.94</b>	6.54	6.10	5.95	5.95	6.14	(2.8)	6.08	(2.8)	<b>5.76</b>	(2.9)	6.41	(2.0)	5.96	(2.3)	5.80	(2.4)	5.80	(2.4)
	Level 0 2 <sup>†</sup>	IAS12	60×1528×8359	5.96	12	6.38	6.27	<b>5.96</b>	6.55	6.11	5.97	5.98	6.21	(2.7)	6.10	(2.8)	<b>5.79</b>	(2.9)	6.43	(1.9)	5.98	(2.2)	5.83	(2.4)	5.85	(2.3)
	Level 0 3 <sup>†</sup>	IAS13	60×1528×8359	5.25	12	6.31	6.19	<b>5.89</b>	6.48	6.04	5.90	5.91	6.14	(2.7)	6.01	(2.9)	<b>5.71</b>	(3.0)	6.35	(2.0)	5.91	(2.2)	5.76	(2.3)	5.77	(2.3)
	Level 0 4 <sup>†</sup>	IAS14	60×1528×8359	6.30	12	6.43	6.37	<b>6.04</b>	6.65	6.20	6.06	6.07	6.25	(2.9)	6.19	(2.8)	<b>5.87</b>	(2.9)	6.52	(2.0)	6.07	(2.2)	5.92	(2.3)	5.91	(2.6)
	Average			5.85		6.36	6.27	<b>5.96</b>	6.56	6.11	5.97	5.98	6.19	(2.8)	6.09	(2.8)	<b>5.78</b>	(2.9)	6.43	(2.0)	5.98	(2.2)	5.83	(2.4)	5.83	(2.4)

†: Uncalibrated (U). ‡: Calibrated (C). \*: S9: Simple-9, S16: Simple-16, PFD: PForDelta.

a fixed interval, and at each interval, the partial sums of the number of integers are computed. With these strategies, random cell access can be done in constant time. Note that, however, it may incur some overhead and this should be taken into consideration when used in a real-time application.

#### IV. EXPERIMENTAL RESULTS

TABLE II

COMPARISON OF RANDOM ACCESS TIME ( $\mu\text{s}$ ) BETWEEN DIFFERENT INTEGER ENCODERS FOR PADDED AND UNPADDED MATRICES. THE BEST RESULTS ARE HIGHLIGHTED. THE ABBREVIATIONS FOR THE INTEGER ENCODERS ARE THE SAME AS IN TABLE I.

Scene Data	Padded Matrix ( $\mu\text{s}$ )							Unpadded Matrix ( $\mu\text{s}$ )						
	DACs	S9	S16	PFD 32	PFD 64	PFD 128	PFD 256	DACs	S9	S16	PFD 32	PFD 64	PFD 128	PFD 256
AG9	0.65	0.65	0.65	0.62	0.59	<b>0.56</b>	0.66	<b>0.58</b>	0.67	0.66	0.60	0.62	0.62	0.70
AG16	0.58	0.67	0.60	0.57	<b>0.56</b>	0.57	0.60	<b>0.55</b>	0.64	0.65	0.58	0.61	0.63	0.71
AG60	0.63	0.67	0.69	<b>0.56</b>	0.61	0.61	0.67	<b>0.55</b>	0.64	0.67	0.59	0.60	0.65	0.73
AG126	0.51	0.69	0.64	0.59	0.50	<b>0.49</b>	0.62	<b>0.60</b>	0.65	0.64	0.62	0.63	0.66	0.73
AG129	0.67	0.67	0.69	<b>0.57</b>	0.58	<b>0.57</b>	0.63	<b>0.58</b>	0.63	0.62	0.59	0.59	0.61	0.70
AG151	0.68	0.64	0.67	<b>0.54</b>	0.57	0.60	0.68	<b>0.60</b>	0.65	0.65	0.61	0.62	0.63	0.72
AG182	0.69	0.67	0.72	<b>0.55</b>	0.61	0.57	0.78	<b>0.67</b>	<b>0.67</b>	0.68	0.75	0.74	0.80	0.89
AG193	0.57	0.65	0.68	0.58	0.57	<b>0.56</b>	0.69	<b>0.55</b>	0.65	0.64	0.60	0.60	0.65	0.70
Average	0.62	0.66	0.67	0.57	0.57	<b>0.56</b>	0.67	<b>0.58</b>	0.65	0.65	0.62	0.63	0.66	0.73
AC00	1.59	1.63	1.63	<b>1.47</b>	1.51	1.67	1.67	0.71	0.77	0.77	<b>0.70</b>	0.71	0.73	0.79
AC03	1.54	1.58	1.62	1.45	<b>1.37</b>	1.47	1.58	<b>0.69</b>	0.77	0.77	0.71	0.71	0.73	0.79
AC10	1.91	1.63	1.65	1.62	1.59	<b>1.34</b>	1.51	0.89	0.76	0.76	<b>0.68</b>	<b>0.68</b>	0.69	0.76
AC11	1.83	1.64	1.63	<b>1.36</b>	1.52	1.49	1.48	0.86	0.77	0.77	<b>0.70</b>	0.71	0.71	0.76
AC18	1.59	1.62	1.64	<b>1.44</b>	1.53	1.45	1.59	0.72	0.78	0.77	<b>0.70</b>	<b>0.70</b>	0.73	0.78
Average	1.69	1.62	1.64	<b>1.47</b>	1.50	1.48	1.56	0.78	0.77	0.77	<b>0.70</b>	<b>0.70</b>	0.72	0.78
AU00	0.93	1.05	1.05	0.99	<b>0.91</b>	1.01	0.92	0.60	0.65	0.66	<b>0.59</b>	<b>0.59</b>	0.61	0.66
AU03	0.92	1.05	1.04	<b>0.84</b>	0.89	0.99	0.92	<b>0.59</b>	0.65	0.65	<b>0.59</b>	<b>0.59</b>	0.59	0.62
AU10	1.24	1.05	1.06	0.92	0.97	<b>0.87</b>	1.01	0.78	0.64	0.64	<b>0.57</b>	<b>0.57</b>	0.57	0.65
AU11	1.13	1.05	1.02	0.95	<b>0.94</b>	<b>0.87</b>	1.05	0.73	0.66	0.65	<b>0.57</b>	<b>0.57</b>	0.58	0.60
AU18	0.99	1.10	1.10	<b>0.88</b>	0.99	0.99	0.98	0.61	0.66	0.66	0.59	<b>0.58</b>	0.59	0.62
Average	1.04	1.06	1.05	<b>0.91</b>	0.94	0.94	0.98	0.66	0.65	0.65	<b>0.58</b>	<b>0.58</b>	0.59	0.63
CR1	1.66	1.64	1.62	<b>1.43</b>	1.52	1.56	1.60	0.82	0.88	0.89	<b>0.78</b>	<b>0.78</b>	0.80	0.87
CR2	1.71	1.60	1.74	1.50	<b>1.41</b>	1.65	1.51	0.85	0.89	0.90	0.81	<b>0.80</b>	0.81	0.89
CR3	1.59	1.72	1.73	1.51	<b>1.49</b>	1.52	1.53	<b>0.80</b>	0.92	0.91	0.81	0.81	0.81	0.87
CR4	1.43	1.49	1.42	1.22	<b>1.20</b>	1.26	1.37	0.82	0.81	0.81	0.74	<b>0.73</b>	0.75	0.83
CR5	1.48	1.51	1.54	<b>1.22</b>	1.26	1.27	1.42	0.84	0.82	0.81	<b>0.73</b>	0.74	0.75	0.84
CR6	1.44	1.50	1.53	1.32	<b>1.29</b>	1.32	1.44	0.77	0.82	0.82	<b>0.73</b>	<b>0.73</b>	0.76	0.83
Average	1.55	1.58	1.60	1.37	<b>1.36</b>	1.43	1.48	0.82	0.86	0.86	0.77	<b>0.76</b>	0.78	0.85
HC1	1.52	1.64	1.63	1.53	<b>1.44</b>	1.49	1.65	<b>0.62</b>	0.71	0.71	0.64	0.65	0.67	0.72
HC2	1.63	1.56	1.49	<b>1.47</b>	1.49	1.48	1.68	0.71	0.70	0.70	0.64	<b>0.63</b>	0.65	0.69
HC3	1.38	1.47	1.50	1.40	1.40	<b>1.36</b>	1.47	<b>0.63</b>	0.70	0.70	0.64	0.64	0.65	0.70
Average	1.51	1.56	1.54	1.47	<b>1.44</b>	1.48	1.56	0.65	0.70	0.70	<b>0.64</b>	<b>0.64</b>	0.65	0.70
HU1	1.91	2.02	2.02	1.70	1.72	<b>1.61</b>	1.76	<b>0.79</b>	0.88	0.89	<b>0.79</b>	<b>0.79</b>	0.80	0.84
HU2	1.82	2.01	2.00	1.76	<b>1.68</b>	1.82	1.85	<b>0.78</b>	0.89	0.88	0.79	0.79	0.81	0.87
HU3	1.82	2.02	2.05	1.82	1.83	<b>1.77</b>	1.87	<b>0.76</b>	0.89	0.89	0.79	0.78	0.80	0.85
Average	1.85	2.02	2.02	1.76	1.74	<b>1.73</b>	1.83	<b>0.78</b>	0.89	0.89	0.79	0.79	0.80	0.85
IASH1	1.34	1.24	1.30	1.10	<b>1.02</b>	1.05	1.09	<b>1.25</b>	1.46	1.32	1.27	1.31	1.38	1.44
IASI2	1.34	1.29	1.27	<b>1.12</b>	<b>1.12</b>	<b>1.12</b>	<b>1.12</b>	<b>1.25</b>	1.47	1.30	1.33	1.30	1.39	1.43
IASI3	1.35	1.31	1.29	1.11	1.12	<b>1.02</b>	1.13	<b>1.29</b>	1.42	1.30	1.31	1.31	1.40	1.45
IASH4	1.38	1.32	1.24	1.11	<b>1.06</b>	1.09	1.14	<b>1.27</b>	1.43	1.34	1.32	1.33	1.35	1.45
Average	1.35	1.29	1.28	1.11	1.08	<b>1.07</b>	1.12	<b>1.27</b>	1.45	1.32	1.31	1.31	1.38	1.44

### B. Random Query With and Without Padding

In this section, we test the time it takes to query elements. Random access to elements in each tested scene is performed in 100 000 iterations with and without padding. The GetCell function in the paper by Ladra *et al.* [11] is modified using partial sums and sampling to optimize random cell access time in an unpadded matrix. To ensure more accurate results, we use the same set of coordinates and bands generated randomly for each scene for all the different encoders and matrices. The program is repeated 20 times for each scene and the average time is taken. The results are shown in Table II. It shows that the best access time is to use the 32-, 64-, and 128-integer PForDelta encoders for padded matrices and DACs, and 64- and 32-integer PForDelta encoders for unpadded matrices. What is more notable is that the access times are cut almost in half (49%) for most of the unpadded matrices, whereas AIRS Granules and IASI data have more or less the same access times for both types of matrices, possibly due to the similar tree data built from the padded and unpadded matrices. Other factors influencing execution times are the effectiveness of the partial sums and sampling optimization, and the relatively small spatial sizes of AIRS and IASI matrices (e.g., AIRS has  $90 \times 135 = 12\,150$  pixels) compared with others (e.g., AVIRIS Yellowstone has  $680 \times 512 = 348\,160$  pixels). Thus, the effects are less noticeable. Also, we should note that for access times in unpadded matrices, the way the tree is traversed is greatly boosted using the partial sums and sampling, which help access elements faster.

These experiments prove that the storage space and random access to elements in the  $k^2$ -raster structure produce competitive results not only for DACs but also for the Simple family of word-aligned integer encoders.

### V. CONCLUSION

In this research, we propose a new no-padding method to reduce the storage space by saving only the elements in the nodes of a  $k^2$ -raster that are within the bounds of the original matrix, and our experiments have shown that it saves space up to 6%. The access time has also been reduced by half for most of the data when using unpadded matrices. Furthermore, the use of other random access integer encoders, such as Simple-9, Simple-16, and PForDelta, has proven to be competitive compared with DACs, the encoder originally used by the authors of  $k^2$ -raster. In particular, we can see that for most hyperspectral data, PForDelta performs better than DACs with up to 6% reduction in storage size and up to 20% reduction in random access time to elements. The experiments also show that the Simple family of integer encoders can also be used as a good alternative to DACs for random access to integer sequences.

### REFERENCES

- [1] G. Navarro, *Compact Data Structures: A Practical Approach*. Cambridge, U.K.: Cambridge Univ. Press, 2016.
- [2] G. Jacobson, "Space-efficient static trees and graphs," in *Proc. 30th Annu. Symp. Found. Comput. Sci.*, Oct. 1989, pp. 549–554.
- [3] *Low-Complexity Lossless and Near-Lossless Multispectral and Hyperspectral Image Compression. Blue Book. Issue 2, Consultative Committee for Space Data Systems (CCSDS)*, Standard CCSDS 123.0-B-2, Feb. 2019. [Online]. Available: <https://public.ccsds.org/Pubs/123x0b2c3.pdf>
- [4] K. Chow, D. Tzamarías, I. Blanes, and J. Serra-Sagrà, "Using predictive and differential methods with  $K^2$ -raster compact data structure for hyperspectral image lossless compression," *Remote Sens.*, vol. 11, no. 21, p. 2461, Oct. 2019.
- [5] N. R. Brisaboa, S. Ladra, and G. Navarro, "DACs: Bringing direct access to variable-length codes," *Inf. Process. Manage.*, vol. 49, no. 1, pp. 392–404, Jan. 2013.
- [6] N. R. Brisaboa, M. R. Luaces, G. Navarro, and D. Seco, "A fun application of compact data structures to indexing geographic data," in *Proc. Int. Conf. Fun With Algorithms*. Berlin, Germany: Springer, 2010, pp. 77–88.
- [7] N. Brisaboa, A. Fariña, G. Navarro, and J. Paramá, "Dynamic lightweight text compression," *ACM Trans. Inf. Syst.*, vol. 28, no. 3, pp. 1–32, Jun. 2010.
- [8] N. R. Brisaboa, S. Ladra, and G. Navarro, "Compact representation of Web graphs with extended functionality," *Inf. Syst.*, vol. 39, pp. 152–174, Jan. 2014.
- [9] N. R. Brisaboa, M. A. Rodríguez, D. Seco, and R. A. Troncoso, "Rank-based strategies for cleaning inconsistent spatial databases," *Int. J. Geograph. Inf. Sci.*, vol. 29, no. 2, pp. 280–304, Feb. 2015.
- [10] F. Silva-Coira, "Compact data structures for large and complex datasets," Ph.D. dissertation, Facultad de Informática Universidad de Coruña, A Coruña, Spain, 2017.
- [11] S. Ladra, J. R. Paramá, and F. Silva-Coira, "Scalable and queryable compressed storage structure for raster data," *Inf. Syst.*, vol. 72, pp. 179–204, Dec. 2017.
- [12] K. Chow, D. E. O. Tzamarías, M. Hernández-Cabronero, I. Blanes, and J. Serra-Sagrà, "Analysis of variable-length codes for integer encoding in hyperspectral data compression with the  $k^2$ -raster compact data structure," *Remote Sens.*, vol. 12, no. 12, p. 1983, Jun. 2020.
- [13] V. N. Anh and A. Moffat, "Inverted index compression using word-aligned binary codes," *Inf. Retr.*, vol. 8, no. 1, pp. 151–166, Jan. 2005.
- [14] J. Zhang, X. Long, and T. Suel, "Performance of compressed inverted list caching in search engines," in *Proc. 17th Int. Conf. World Wide Web (WWW)*, 2008, pp. 387–396.
- [15] M. Zukowski, S. Heman, N. Nes, and P. Boncz, "Super-scalar RAM-CPU cache compression," in *Proc. 22nd Int. Conf. Data Eng. (ICDE)*, Apr. 2006, p. 59.

## Chapter 5

**A compact data structure for  
hyperspectral scenes based on  
raster time series**

# A Compact Data Structure for Hyperspectral Scenes Based on Raster Time Series

Kevin Chow, Dion Eustathios Olivier Tzarmarias, Miguel Hernández-Cabronero, Ian Blanes, Diego Seco and Joan Serra-Sagrà, *Senior Member, IEEE*

**Abstract**—In this letter, the state-of-the-art T- $k^2$ raster, a compact data structure, is examined for lossless hyperspectral data compression. This structure is a combination of  $k^2$ raster and a modified version of  $k^2$ raster, both of which are  $k^2$ -ary tree data structures that provide efficient storage and real-time processing. T- $k^2$ raster, which in earlier research produced favorable results for raster time series, is now being studied for remote sensing data. The structure can be used to minimize redundancies in hyperspectral data resulting from the spectral correlation between elements in adjacent bands. In this research, T- $k^2$ raster has been used with hyperspectral scenes obtained from real missions to determine whether it can provide favorable compression ratios and random element access. In previous experiments,  $k^2$ raster helped reduce the hyperspectral data up to 52% compared to the uncompressed data. In this work, it is shown that compression with T- $k^2$ raster provides further size reduction for our test data, with up to 19% decrease in size compared to  $k^2$ raster or 61% compared to the uncompressed data. The data access time is faster in most cases, with up to 25% of speedup.

**Index Terms**—lossless image compression, hyperspectral imaging, remote sensing, compact data structures,  $k^2$ raster, T- $k^2$ raster, directly addressable codes.

## I. INTRODUCTION

**H**YPERSPECTRAL scenes are image data composed of various numbers of bands from across the electromagnetic spectrum. They are taken by airborne sensors such as Airborne Visible/Infrared Imaging Spectrometer (AVIRIS), or satellites in space such as Hyperion or Infrared Atmospheric Sounding Interferometer (IASI). The hyperspectral scenes need to be compressed to allow for effective transmission under the constraining limits of data links and for minimizing storage space [1].

Compact data structures (CDS) are structures that store data efficiently (i.e., data compression is actually achieved) while providing real-time access to data in the compressed

domain [2]. They are usually loaded into memory and with the rank and select functions [3] as their main primitive operations, they can perform operations such as random access to elements. The most notable advantage of using a CDS is that, in order to access or query individual data, there is no need to decompress the whole CDS structure. By contrast, data compressed by classical compression algorithms such as Gzip, and specialized algorithms such as CCSDS 123.0-B-2 [4], [5] need to be fully decompressed before data can be accessed and used. Besides image compression, CDS can also be used for text compression such as FM-Index [6] and the Burrows-Wheeler Transform [7]. They are used in different areas such as sequence alignment [8] and medical imaging [9]. A CDS proposed by researchers at the Universidade da Coruña known as  $k^2$ -tree [10] is used in compressing web graph representation that takes advantage of large empty areas of adjacency matrix of the graph.  $k^2$ -tree built from web graphs contains nodes which are binary bits.

In previous works, we have made use of a CDS called  $k^2$ raster [11], which is similar to  $k^2$ -tree but has nodes that hold integer values instead of binary bits. Predictive and differential methods for  $k^2$ raster were developed to take advantage of the high redundancies between spectral bands in hyperspectral data, and these two methods produced favorable results [12]. The downside of using them, however, is that they only provide random access to individual cells while other operations such as query on a region cannot be done. Another research work was conducted to analyze the performance gains of using different integer encoders such as Rice codes [13], Simple-9 [14], Simple-16 [15] and PForDelta [16] instead of the Directly Addressable Codes (DACs), originally used by  $k^2$ raster. In [17] and [18], it was shown that these encoders provided competitive results.

Raster time series or temporal rasters are collections of rasters covering the same region at consecutive timestamps. A new space-efficient representation of raster time series, T- $k^2$ raster, based on CDS was proposed by Silva-Coira et al. [11]. This structure takes advantage of the temporal regularities between consecutive rasters in a time series. It uses a strategy of snapshots and logs to represent the data. There are two versions, one using regular sampling of timestamps and the other being based on a heuristic where the sampling is irregular.

In this work, instead of raster time series, hyperspectral data is used in our study of T- $k^2$ raster in order to take advantage of the spectral correlation between neighboring

K. Chow, D.E.O. Tzarmarias, M. Hernández-Cabronero, I. Blanes, and J. Serra-Sagrà are with the Department of Information and Communications Engineering, Universitat Autònoma de Barcelona, Cerdanyola del Vallès, 08193 Spain (e-mail: kevin.chow@uab.cat).

D. Seco is with the Department of Computer Science and Information Technologies, Universidade de A Coruña, Spain.

This research was funded by the Spanish Ministry of Economy and Competitiveness, the Spanish Ministry of Science and Innovation, and the European Regional Development Fund under Grants RTI2018-095287-B-I00, PID2021-125258OB-I00 and BES-2016-078369 (Programa Formación de Personal Investigador), by the Catalan Government under Grant 2017SGR-463, by the postdoctoral fellowship program Beatriu de Pinós, Reference 2018-BP-00008, funded by the Secretary of Universities and Research (Government of Catalonia), and by the Horizon 2020 program of research and innovation of the European Union under the Marie Skłodowska-Curie Grant Agreement #801370.



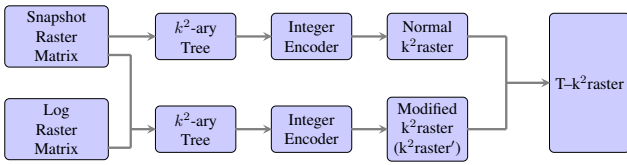


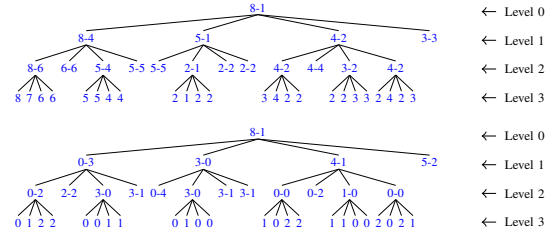
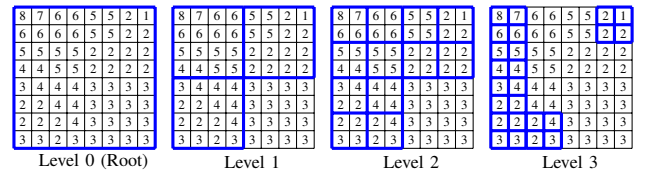
Fig. 1: Construction of T- $k^2$ -raster. A  $k^2$ -ary tree is first built from a snapshot raster matrix and/or log raster matrix. Compression and random access are achieved when tree node data are encoded by an integer encoder such as DACs, Simple-16, or PForDelta, producing a  $k^2$ -raster or a  $k^2$ -raster' structure. Finally, these two structures are combined to form a T- $k^2$ -raster structure.

bands. We carry out experiments for the two versions of T- $k^2$ -raster with the purpose of quantifying the storage size and access time improvements for hyperspectral image compression. Fig. 1 shows the interrelations between the different elements in forming the T- $k^2$ -raster structure.

The rest of the paper is organized as follows: Section II provides background information on  $k^2$ -raster and T- $k^2$ -raster. Section III discusses in detail how T- $k^2$ -raster and heuristic T- $k^2$ -raster work. In section IV, experimental results are reported and analyzed. Finally, we conclude with a summary in Section V.

## II. BACKGROUND

Ladra et al. proposed  $k^2$ -raster, a tree structure specifically designed for raster data including images [19]. It is built from a matrix of width  $w$  and height  $h$ , and an integer  $k \geq 2$ . If the matrix can be partitioned into  $k^2$  square quadrants of equal size, it can be used directly. Otherwise, it is necessary to enlarge the matrix to size  $s \times s$ , where  $s = k^{\lceil \log_k \max(w, h) \rceil}$ , and the number of subdivisions is  $\log_k(s)$ . The padding elements are equal to zero. This extended matrix is then recursively partitioned into  $k^2$  square submatrices of identical size, hereafter referred to as quadrants. This process is repeated until all cells in a quadrant have the same value, or until the submatrix has size  $1 \times 1$ . This partitioning produces the nodes for a  $k^2$ -ary tree topology where the data in the nodes is stored in the following data structures: First, at each tree level  $\ell$ , the maximum and minimum values of each quadrant are computed. These are then compared with the corresponding maximum and minimum values in the parent node, and the differences are stored in the  $V_{max_\ell}$  and  $V_{min_\ell}$  arrays of each level. Saving the differences instead of the original values results in smaller values for each node, which in turn allows a better compression with an integer encoder. Therefore,  $k^2$ -raster helps reduce the size of hyperspectral data by taking advantage of its spatial correlation. Next, with the exception of the root node at the top level, the  $V_{max_\ell}$  and  $V_{min_\ell}$  arrays at all the levels are concatenated to form the  $L_{max}$  and  $L_{min}$  arrays, respectively. Both arrays are then compressed by an integer encoder. Second, the root's maximum and minimum values are stored as uncompressed integers. They become the first element of, respectively,  $L_{max}$  and  $L_{min}$ . Third, a bitmap array  $T$  is generated from all the nodes except the ones at the root and at the last level,



Element	Base	Tree Level	Node Data
$L_{max}$	10	Level 0	8
		Level 1	0345
		Level 2	0233 0333 0010
		Level 3	0122 0011 0100 1022 1100 2021
$L_{min}$	10	Level 0	1
		Level 1	301
		Level 2	20 0 000
$T$ Bitmap	2		1 1110 1010 0100 1011

Fig. 2: (Top) A snapshot matrix ( $M_S$ ) showing recursive partitioning. (Middle) The upper tree is a  $k^2$ -raster ( $k = 2$ ) tree constructed from the matrix and the lower tree takes into account the differences between the parent and child nodes. Maximum and minimum values in each node are separated by a hyphen. (Bottom) A table showing the elements of the  $k^2$ -raster.

each node indicating whether it has child nodes or not. This bitmap serves to locate the tree nodes when cell queries are performed by means of a rank function [3]. In Fig. 2, an example of an  $8 \times 8$  matrix is shown to illustrate this process. The elements which fully describe the resulting  $k^2$ -raster structure are shown at the bottom of Fig. 2. Please refer to [18] for a more comprehensive description of  $k^2$ -raster.

The Directly Addressable Codes (DACs) were used as the integer encoder for the compression of  $L_{max}$  and  $L_{min}$  in the original paper for  $k^2$ -raster. For the experiments in this research, DACs and other integer encoders such as PForDelta and Simple-16 were used. For more information on DACs, please refer to the paper by Ladra et al. [20]. For a discussion of how  $k^2$ -raster performs in tandem with different integer encoders, interested readers can refer to our previous papers [17], [18].

## III. PROPOSED T- $k^2$ -RASTER AND HEURISTIC T- $k^2$ -RASTER

### A. T- $k^2$ -raster

The raster time series T- $k^2$ -raster was proposed by Silva-Coira et al. [11] and was originally designed to take advantage of the temporal regularities between consecutive rasters in a time series. This can also be extended to other 3-D data such as remote sensing data where raster elements in each band are likely to have high redundancies with the corresponding ones in the neighboring spectral bands. The T- $k^2$ -raster structure is built by regularly grouping the raster matrices that contain a combination of *snapshots* and *logs*, where the group size is denoted by  $t_\delta$  and  $t_\delta \geq 2$ .

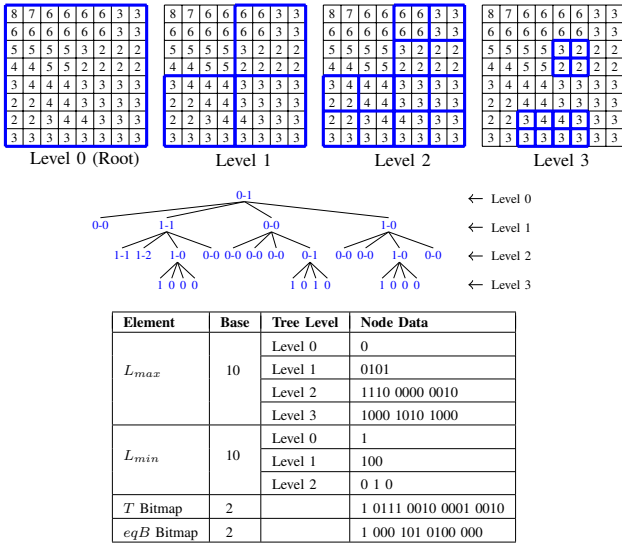


Fig. 3: (Top) A non-snapshot matrix ( $M_{s+1}$ ) showing recursive partitioning. (Middle) a T- $k^2$ raster ( $k = 2$ ) tree constructed from the differences between the matrix ( $M_{s+1}$ ) at the top of this figure and the matrix in Fig. 2 ( $M_s$ ). Except for the last level, all nodes show the maximum and minimum values separated by a hyphen. (Bottom) A table showing the elements of the T- $k^2$ raster.

This means that in each group, the first raster ( $M_s$ ) is the snapshot raster and subsequent rasters ( $M_{s+i}$ ) are the log rasters. Elements in these log rasters ( $M_{s+i}$ ) are compared to those corresponding elements in the first raster ( $M_s$ ) and the differences are taken, and as a result a modified  $k^2$ raster' is created. For ease of discussion, a  $k^2$ raster-built band is denoted by  $S$  and a  $k^2$ raster'-built band by  $P$ . A sequence lists the type of bands,  $S$  or  $P$ , that make up the T- $k^2$ raster structure from the first band to the last.

At the top of both Figs. 2 and 3, a snapshot raster matrix ( $M_s$ ) and a log raster matrix ( $M_{s+1}$ ) in the next band are shown respectively. To build a T- $k^2$ raster from  $M_{s+1}$ , the values of the nodes in the tree in the middle of Fig. 3 are computed by taking the spectral differences between these two rasters. This means that in the process of recursive partitioning of  $M_{s+1}$ , the elements in each submatrix is compared to the corresponding elements in  $M_s$ . If they are the same, the partitioning process will stop and this particular node becomes a leaf in the tree structure for  $M_{s+1}$ . Doing so will save space by minimizing the redundancies that exist between the snapshot and log raster matrices. To better understand, one can compare this with  $k^2$ raster where spatial correlation between neighboring elements is exploited by taking the differences between the parent and child tree nodes in the same band. However, this regularity may not necessarily provide the best compression and further improvement in size can be made by building some or all of these log matrices as normal  $k^2$ rasters. In other words, in each group, the first raster will always be an  $S$  band while the rest could be an  $S$  or  $P$  band.

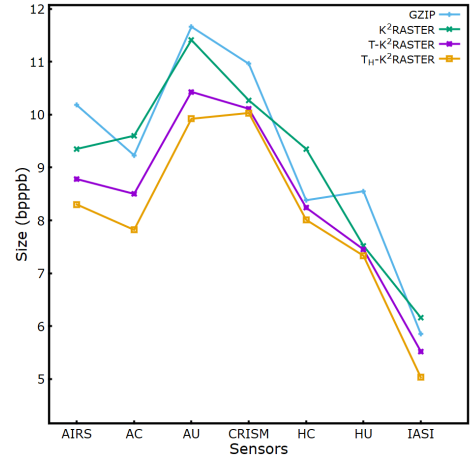


Fig. 4: A bit-rate (bpppb) comparison of storage size between different compression structures for 7 different datasets with DACs as integer encoder.

### B. Heuristic T- $k^2$ raster

A heuristic approach to T- $k^2$ raster (or  $T_H$ - $k^2$ raster) where further improvement can be made is discussed in [11]. The data is analyzed one band at a time. To illustrate with an example, we start off by building the first band as a normal  $k^2$ raster and the band sequence begins thus: ( $S$ ). Then the second band is built either as an  $S$  band or as a  $P$  band with respect to the first band. The one with a smaller size will be used, resulting in either ( $S, S$ ) or ( $S, P$ ). For this example, we assume the second band is a  $P$  band and the sequence is ( $S, P$ ). For the third band, the same procedure is repeated so that whether to use an  $S$  band or a  $P$  band is based upon the size produced. If having an  $S$  band produces a smaller size, then it will be used and the sequence becomes ( $S, P, S$ ). Otherwise, a  $P$  band (built with respect to the first  $S$  band) is used and the sequence becomes ( $S, P, P$ ). At this point, there is one more testing that can be performed for even more improvement. That is, we can look back at the second band and rebuild this band as  $k^2$ raster and then rebuild the third band as  $k^2$ raster' with respect to this new  $k^2$ raster structure in the second band. If these two structures combined together produce a smaller size, then they will be used instead and the resulting sequence will be ( $S, S, P$ ). This process aimed at achieving smaller sizes is repeated for each band until it reaches the last one. In this heuristic approach, the grouping of the bands will be irregular and  $t_\delta \geq 1$ . In the experimental results section, we can see that this approach has proven to produce the best storage size for all our test data.

For a more detailed discussion on T- $k^2$ raster and  $T_H$ - $k^2$ raster, please refer to the original paper where the algorithms for building these structures (build( $\cdot$ )) and for accessing elements in these structures (getCell( $\cdot$ )) can be found [11].

## IV. EXPERIMENTAL RESULTS

This section presents some of the experimental results that were conducted using  $k^2$ raster, T- $k^2$ raster, and  $T_H$ - $k^2$ raster.

TABLE I: Testing scene data used and a comparison of  $k^2$ raster storage size and random access time with those of  $T_H$ - $k^2$ raster. Storage size reductions (%) from  $k^2$ raster to  $T_H$ - $k^2$ raster for different integer encoders are enclosed in parentheses in the  $T_H$ - $k^2$ raster column. Similarly, changes in random access time (%) are also enclosed in parenthesis. In the dimensions column,  $x$  is the image width,  $y$  the image height and  $z$  the number of spectral bands. The best results are highlighted and underscored.

Sensor	Name	Acronym	Original Dimensions (x × y × z)	Best k Value	Storage size (bpppb)									Random access time		
					Gzip	DACs			Simple-16			PForDelta-128			DACs	
						$k^2$ raster	$T$ - $k^2$ raster ( $t_\delta = 2$ )	$T_H$ - $k^2$ raster	$k^2$ raster	$T$ - $k^2$ raster ( $t_\delta = 2$ )	$T_H$ - $k^2$ raster	$k^2$ raster	$T$ - $k^2$ raster ( $t_\delta = 2$ )	$T_H$ - $k^2$ raster	$k^2$ raster	$T$ - $k^2$ raster
AIRS	9 <sup>†</sup>	AG9	90×135×1501	6	10.16	9.42	8.75	8.18 (13.2)	9.63	8.89	8.22 (14.6)	9.53	8.71	<u>8.04</u> (15.6)	0.53	<u>0.48</u> (9.3)
	16 <sup>†</sup>	AG16	90×135×1501	6	9.82	9.05	8.43	7.90 (12.7)	9.24	8.57	7.90 (14.5)	9.09	8.38	<u>7.77</u> (14.5)	0.50	<u>0.49</u> (2.6)
	60 <sup>†</sup>	AG60	90×135×1501	6	10.53	9.74	9.08	8.56 (12.1)	10.05	9.32	8.65 (13.9)	9.68	8.93	<u>8.30</u> (14.3)	0.52	<u>0.48</u> (6.7)
	126 <sup>†</sup>	AG126	90×135×1501	6	10.33	9.54	8.93	8.46 (11.3)	9.75	9.11	8.55 (12.3)	9.47	8.80	<u>8.23</u> (13.1)	0.57	<u>0.48</u> (15.1)
	129 <sup>†</sup>	AG129	90×135×1501	6	9.50	8.58	8.13	7.72 (10.0)	8.55	8.11	7.66 (10.4)	8.61	8.06	<u>7.59</u> (11.8)	0.56	<u>0.51</u> (7.7)
	151 <sup>†</sup>	AG151	90×135×1501	6	10.31	9.46	8.98	8.58 (9.3)	9.48	9.01	8.58 (9.5)	9.55	8.78	<u>8.30</u> (11.2)	0.59	<u>0.54</u> (7.9)
	182 <sup>†</sup>	AG182	90×135×1501	6	10.64	9.62	9.10	8.64 (10.2)	9.95	9.31	8.73 (12.3)	9.70	9.01	<u>8.44</u> (13.0)	0.59	<u>0.58</u> (2.0)
	193 <sup>†</sup>	AG193	90×135×1501	6	10.15	9.37	8.80	8.34 (11.0)	9.59	8.98	8.46 (11.8)	9.59	8.73	<u>8.19</u> (12.8)	0.53	<u>0.52</u> (2.1)
	Average				10.18	9.35	8.78	8.30 (11.2)	9.53	8.91	8.34 (12.4)	9.55	8.68	<u>8.11</u> (13.3)	0.55	<u>0.51</u> (6.7)
AVIRIS	Yellowstone sc. 00 <sup>‡</sup>	AC00	677×512×224	6	10.12	9.81	8.86	8.22 (16.2)	10.67	9.34	8.38 (21.5)	9.75	8.67	<u>8.02</u> (17.7)	1.36	<u>0.99</u> (27.3)
	Yellowstone sc. 03 <sup>‡</sup>	AC03	677×512×224	6	9.59	9.92	8.80	<u>8.07</u> (18.6)	10.82	9.31	8.34 (22.9)	9.81	8.74	8.13 (17.1)	1.34	<u>1.00</u> (25.2)
	Yellowstone sc. 10 <sup>‡</sup>	AC10	677×512×224	6	7.41	9.00	7.93	<u>7.26</u> (19.3)	10.31	8.69	7.69 (25.4)	8.97	8.19	7.71 (14.0)	1.58	<u>1.25</u> (21.1)
	Yellowstone sc. 11 <sup>‡</sup>	AC11	677×512×224	6	9.04	9.22	8.17	<u>7.50</u> (18.7)	10.09	8.70	7.64 (24.3)	9.23	8.23	7.54 (18.3)	1.61	<u>1.18</u> (26.3)
	Yellowstone sc. 18 <sup>‡</sup>	AC18	677×512×224	6	10.00	10.04	8.74	<u>8.06</u> (19.7)	10.99	9.31	8.34 (24.1)	9.90	8.67	<u>7.99</u> (19.3)	1.33	<u>0.97</u> (26.8)
	Average				9.23	9.60	8.50	<u>7.82</u> (18.5)	10.58	9.07	8.08 (23.6)	9.53	8.50	7.88 (17.4)	1.44	<u>1.08</u> (25.3)
	Yellowstone sc. 00 <sup>†</sup>	AU00	680×512×224	9	12.39	11.92	10.97	10.45 (12.3)	13.79	12.44	11.67 (15.4)	11.44	10.45	<u>9.99</u> (12.7)	0.83	<u>0.61</u> (26.1)
	Yellowstone sc. 03 <sup>†</sup>	AU03	680×512×224	9	11.98	11.74	10.72	10.18 (13.2)	13.29	11.94	11.16 (16.0)	11.08	10.12	<u>9.66</u> (12.8)	0.81	<u>0.60</u> (26.7)
	Yellowstone sc. 10 <sup>†</sup>	AU10	680×512×224	9	10.17	9.98	9.36	9.01 (9.7)	10.53	9.78	9.30 (11.7)	9.10	8.60	<u>8.34</u> (8.4)	1.10	<u>0.90</u> (18.2)
Yellowstone sc. 11 <sup>†</sup>	AU11	680×512×224	9	11.49	11.27	10.22	9.65 (14.4)	12.89	11.48	10.61 (17.7)	10.77	9.79	<u>9.30</u> (13.6)	1.02	<u>0.86</u> (16.0)	
Yellowstone sc. 18 <sup>†</sup>	AU18	680×512×224	9	12.29	12.15	10.88	10.29 (15.3)	14.01	12.31	11.46 (18.2)	11.62	10.43	<u>9.96</u> (14.3)	0.92	<u>0.61</u> (26.5)	
Average				11.66	11.41	10.43	9.92 (13.1)	12.90	11.59	10.84 (16.0)	10.80	9.88	<u>9.45</u> (12.5)	0.92	<u>0.72</u> (22.1)	
CRISM	frt000065e6_07_sc164 <sup>†</sup>	CR1	640×420×545	6	10.98	9.93	9.80	<u>9.74</u> (1.9)	11.00	10.98	10.97 (0.3)	10.23	10.22	10.22 (0.1)	1.34	<u>1.07</u> (20.3)
	frt00008849_07_sc165 <sup>†</sup>	CR2	640×420×545	6	11.03	10.23	10.11	<u>10.04</u> (1.9)	11.44	11.42	11.39 (0.4)	10.52	10.51	10.51 (0.1)	1.32	<u>0.96</u> (27.1)
	frt0001077d_07_sc166 <sup>†</sup>	CR3	640×480×545	6	11.20	10.93	10.77	<u>10.71</u> (2.0)	12.62	12.56	12.50 (1.0)	11.23	11.22	11.22 (0.1)	1.24	<u>0.80</u> (35.1)
	hrf00004f38_07_sc181 <sup>†</sup>	CR4	320×420×545	5	10.77	9.89	9.61	<u>9.46</u> (4.3)	10.64	10.48	10.39 (2.3)	10.29	10.15	10.06 (2.2)	1.15	<u>1.00</u> (12.5)
	hrf0000648f_07_sc182 <sup>†</sup>	CR5	320×450×545	5	10.90	10.04	9.86	<u>9.76</u> (2.8)	10.92	10.87	10.83 (0.8)	10.47	10.42	10.38 (0.8)	1.22	<u>1.04</u> (14.6)
	hrf0000ba9c_07_sc183 <sup>†</sup>	CR6	320×480×545	5	10.87	10.59	10.51	<u>10.46</u> (1.2)	11.99	11.97	11.95 (0.3)	10.98	10.97	10.98 (0.2)	1.17	<u>0.98</u> (16.2)
Average				10.96	10.27	10.11	<u>10.03</u> (2.3)	11.44	11.38	11.34 (0.8)	10.62	10.58	10.56 (0.6)	1.24	<u>0.98</u> (21.0)	
Hyperion	Agricultural <sup>‡</sup>	HC1	256×3129×242	8	8.84	8.64	7.92	<u>7.69</u> (11.0)	10.47	9.28	8.91 (15.1)	8.81	8.20	8.02 (9.0)	1.37	<u>1.08</u> (21.2)
	Coral Reef <sup>‡</sup>	HC2	256×3127×242	8	7.45	10.03	8.45	<u>8.25</u> (17.7)	14.95	11.57	11.09 (25.8)	10.53	10.04	10.00 (5.0)	1.31	<u>0.94</u> (28.5)
	Urban <sup>‡</sup>	HC3	256×2905×242	8	8.85	9.37	8.34	<u>8.10</u> (13.6)	12.05	10.15	9.64 (20.0)	9.44	8.70	8.53 (9.7)	1.29	<u>1.02</u> (21.0)
	Average				8.38	9.35	8.24	<u>8.01</u> (14.3)	12.49	10.33	9.88 (20.9)	9.59	8.98	8.85 (7.7)	1.32	<u>1.01</u> (23.5)
	Erta Ale <sup>†</sup>	HU1	256×3187×242	8	8.69	7.45	7.39	7.29 (2.1)	7.69	7.63	7.52 (2.3)	7.03	6.98	<u>6.91</u> (1.7)	1.55	<u>1.34</u> (13.5)
Lake Montana <sup>†</sup>	HU2	256×3176×242	8	8.69	7.50	7.45	7.38 (1.6)	7.78	7.73	7.66 (1.6)	7.16	7.12	<u>7.07</u> (1.3)	1.49	<u>1.32</u> (11.6)	
Mt. St. Helena <sup>†</sup>	HU3	256×3242×242	8	8.26	7.60	7.50	7.32 (3.7)	7.90	7.81	7.61 (3.7)	7.21	7.13	<u>7.00</u> (2.9)	1.50	<u>1.23</u> (18.0)	
Average				8.55	7.52	7.45	7.33 (2.5)	7.79	7.72	7.60 (2.5)	7.13	7.08	<u>6.99</u> (2.0)	1.51	<u>1.30</u> (14.4)	
IASI	Level 0 1 <sup>†</sup>	IASI1	60×1528×8359	12	5.90	6.11	5.50	5.03 (17.7)	5.73	5.17	<u>4.75</u> (17.1)	5.74	5.23	4.87 (15.2)	1.22	1.31 (-7.1)
	Level 0 2 <sup>†</sup>	IASI2	60×1528×8359	12	5.96	6.18	5.50	5.00 (19.1)	5.76	5.15	<u>4.71</u> (18.2)	5.77	5.22	5.00 (13.3)	1.24	1.32 (-6.5)
	Level 0 3 <sup>†</sup>	IASI3	60×1528×8359	12	5.25	6.11	5.48	4.99 (18.3)	5.68	5.12	<u>4.71</u> (17.1)	5.69	5.19	4.99 (12.3)	1.27	1.38 (-8.8)
	Level 0 4 <sup>†</sup>	IASI4	60×1528×8359	12	6.30	6.22	5.58	5.10 (18.0)	5.84	5.25	<u>4.81</u> (17.6)	5.85	5.32	5.10 (12.8)	1.25	1.35 (-7.4)
	Average				5.85	6.16	5.52	5.03 (18.3)	5.75	5.17	<u>4.75</u> (17.5)	5.76	5.24	4.99 (13.4)	1.24	1.34 (-7.5)

†: Uncalibrated. ‡: Calibrated.

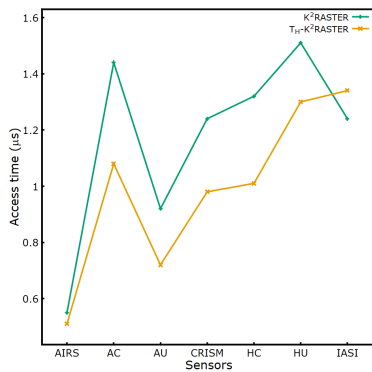


Fig. 5: Access time to element ( $\mu$ s) comparison between  $k^2$ raster and  $T_H$ - $k^2$ raster for 7 different datasets with DACs as integer encoder.

The hyperspectral scenes captured by different sensors in real missions were used. They include: Atmospheric Infrared Sounder (AIRS), AVIRIS, Compact Reconnaissance Imaging Spectrometer for Mars (CRISM), Hyperion, and IASI. They are often used in remote sensing data compression literature and are publicly available (<http://cwe.ccsds.org/sls/docs/sls-dc/123.0-B-Info/TestData>) except for IASI data<sup>1</sup>. These testing data are listed in Table I. Note that the storage size is measured in terms of bits per pixel per band (bpppb).

<sup>1</sup>Available to members of the Consultative Committee for Space Data Systems (CCSDS) only.

Written in C and C++, our implementations of building  $k^2$ raster and the two versions of  $T$ - $k^2$ raster are based, respectively, on the algorithms presented in the paper by Ladra et al. [19] and by Silva-Coira et al. [11]. So is the GetCell(.) function for timing random cell access in both structures. Our software is available on the following website: <https://gici.uab.cat/GiciWebPage/downloads.php#tk2-raster>. The DACs software can be found on the Universidade da Coruña's Database Laboratory website (<http://lbd.udc.es/research/DACS/>). The sources for Simple-16, and PForDelta are available for download from the website for the Poly-IR-Toolkit (<https://code.google.com/archive/p/poly-ir-toolkit/source/default/source>).

The computer used for testing had an Intel(R) Xeon(R) 8-core CPU E3-1230 V2 @ 3.30GHz with 8 MB of cache and 32GB of RAM. The operating system was Ubuntu 20.04 LTS with kernel Linux 5.4.0-26-generic.

#### A. Storage size

For the performance with respect to storage size, we first tested all the data with  $T$ - $k^2$ raster for regular groupings. In other words, the bands were divided into groups of size  $t_\delta$ , and each group consists of the first band being built with  $k^2$ raster and the second band with either  $k^2$ raster or  $k^2$ raster', whichever producing a smaller storage size. The results are listed in Table I under the column  $T$ - $k^2$ raster. Since similar storage sizes were obtained with  $t_\delta \geq 2$  and



sizes get larger with higher  $t_\delta$  values, only  $t_\delta = 2$  is shown in the results. Next, we tested the data using  $T_H$ - $k^2$ -raster. The groupings in this method are irregular. The results are shown under the column  $T_H$ - $k^2$ -raster in Table I. DACs, Simple-16 and PForDelta-128, the integer encoders that produced the best results in our previous paper have been used in  $k^2$ -raster,  $T$ - $k^2$ -raster, and  $T_H$ - $k^2$ -raster.

From Table I and the chart in Fig. 4, it can be seen that  $T_H$ - $k^2$ -raster gives the best results for all the data, followed by  $T$ - $k^2$ -raster and then  $k^2$ -raster. Gzip is shown here for comparison and in general has the largest (worst) compressed sizes compared to the other three structures. For  $T_H$ - $k^2$ -raster, the best results for most data are produced by using DACs or PForDelta-128, while Simple-16 gives the best results when used with IASI data. These experiments have shown that because both  $T$ - $k^2$ -raster and  $T_H$ - $k^2$ -raster can make a choice between spatial differences and spectral differences when building the structure, it will provide better compression for each raster band. For that reason,  $T$ - $k^2$ -raster will always perform better than  $k^2$ -raster, which only exploits the spatial redundancy.

#### B. Access time

We compared the time to access an element in the hyperspectral data between  $k^2$ -raster and  $T_H$ - $k^2$ -raster using DACs as the integer encoder. For each tested scene, data access to a million randomly selected element locations is performed. The average time it takes to access an individual data (in  $\mu$ s) using both structures is shown in Table I and depicted in the chart in Fig. 5.

The results show that there is some improvement in access time to elements, despite the process to access them actually becoming a bit more complicated. This can be explained by the fact that the overall structure in  $T_H$ - $k^2$ -raster has become smaller, thus making it faster to query elements from the DACs-compressed data. For IASI data, results for  $T_H$ - $k^2$ -raster are only 0.1  $\mu$ s slower than for  $k^2$ -raster.

### V. CONCLUSIONS

In this research work, we examined the recently proposed  $T$ - $k^2$ -raster, which is a combination of  $k^2$ -raster and a modified form of  $k^2$ -raster that provides both good compression ratios and access time for hyperspectral data. There are two versions for  $T$ - $k^2$ -raster, one being a normal version, where the bands are regularly grouped, and the other being a heuristic one, where the bands are irregularly grouped.

The experiments show that the storage size in  $T_H$ - $k^2$ -raster has always improved across the board in our hyperspectral data, up to 19% from  $k^2$ -raster. PForDelta-128 and DACs produced the best results when used as the integer encoder for this structure. When considering the random access time,  $T_H$ - $k^2$ -raster has improved as compared with  $k^2$ -raster for most of the sensors by an amount of up to 25%.

For future work, the following ideas can be considered. Firstly, unlike raster time series where the order of the bands is fixed because of the temporal dimension, in hyperspectral data, it is not common to perform queries that expand more

than one raster. Hence, we could reorder the rasters to obtain better performance. To find the optimal order of the rasters is an open problem. Secondly,  $T_H$ - $k^2$ -raster can be formalized as an optimization problem and better heuristics may be found to improve the compression ratio.

### REFERENCES

- [1] I. Blanes, E. Magli, and J. Serra-Sagrasta, "A tutorial on image compression for optical space imaging systems," *IEEE Geoscience and Remote Sensing Magazine*, vol. 2, no. 3, pp. 8–26, 2014.
- [2] G. Navarro, *Compact data structures: A practical approach*. Cambridge University Press, 2016.
- [3] G. Jacobson, "Space-efficient static trees and graphs," in *30th Annual Symposium on Foundations of Computer Science*. IEEE, 1989, pp. 549–554.
- [4] *Low-Complexity Lossless and Near-Lossless Multispectral and Hyperspectral Image Compression. Blue Book. Issue 2*, Consultative Committee for Space Data Systems (CCSDS) Std. CCSDS 123.0-B-2, Feb. 2019. [Online]. Available: <https://public.ccsds.org/Pubs/123x0b2c3.pdf>
- [5] M. Hernandez-Cabronero, A. B. Kiely, M. Klimesh, I. Blanes, J. Ligo, E. Magli, and J. Serra-Sagrasta, "The CCSDS 123.0-B-2 low-complexity lossless and near-lossless multispectral and hyperspectral image compression standard: A comprehensive review," *IEEE Geoscience and Remote Sensing Magazine*, 2021.
- [6] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proceedings 41st annual symposium on foundations of computer science*. IEEE, 2000, pp. 390–398.
- [7] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm," in *Digital SRC Research Report*. Citeseer, 1994.
- [8] J. R. Wang, J. Holt, L. McMillan, and C. D. Jones, "Fmlrc: Hybrid long read error correction using an fm-index," *BMC bioinformatics*, vol. 19, no. 1, pp. 1–11, 2018.
- [9] C. Preston, Z. Arnavut, and B. Koc, "Lossless compression of medical images using burrows-wheeler transformation with inversion coder," in *2015 37th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*. IEEE, 2015, pp. 2956–2959.
- [10] N. R. Brisaboa, S. Ladra, and G. Navarro, " $k^2$ -trees for compact web graph representation," in *International Symposium on String Processing and Information Retrieval*. Springer, 2009, pp. 18–30.
- [11] F. Silva-Coira, J. R. Parama, G. de Bernardo, and D. Seco, "Space-efficient representations of raster time series," *Information Sciences*, vol. 566, pp. 300–325, 2021.
- [12] K. Chow, D. E. O. Tzamarías, I. Blanes, and J. Serra-Sagrasta, "Using predictive and differential methods with  $k^2$ -raster compact data structure for hyperspectral image lossless compression," *Remote Sensing*, vol. 11, no. 21, p. 2461, 2019.
- [13] R. Rice and J. Plaunt, "Adaptive variable-length coding for efficient compression of spacecraft television data," *IEEE Transactions on Communication Technology*, vol. 19, no. 6, pp. 889–897, 1971.
- [14] V. N. Anh and A. Moffat, "Inverted index compression using word-aligned binary codes," *Information Retrieval*, vol. 8, no. 1, pp. 151–166, 2005.
- [15] J. Zhang, X. Long, and T. Suel, "Performance of compressed inverted list caching in search engines," in *Proceedings of the 17th international conference on World Wide Web*, 2008, pp. 387–396.
- [16] M. Zukowski, S. Heman, N. Nes, and P. Boncz, "Super-scalar RAM-CPU cache compression," in *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 2006, pp. 59–59.
- [17] K. Chow, D. E. O. Tzamarías, M. Hernández-Cabronero, I. Blanes, and J. Serra-Sagrasta, "Performance improvement on  $k^2$ -raster compact data structure for hyperspectral scenes," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2022.
- [18] K. Chow, D. E. O. Tzamarías, M. Hernández-Cabronero, I. Blanes, and J. Serra-Sagrasta, "Analysis of variable-length codes for integer encoding in hyperspectral data compression with the  $k^2$ -raster compact data structure," *Remote Sensing*, vol. 12, no. 12, p. 1983, 2020.
- [19] S. Ladra, J. R. Paramá, and F. Silva-Coira, "Scalable and queryable compressed storage structure for raster data," *Information Systems*, vol. 72, pp. 179–204, December 2017.
- [20] N. R. Brisaboa, S. Ladra, and G. Navarro, "DACs: Bringing direct access to variable-length codes," *Information processing & management*, vol. 49, no. 1, pp. 392–404, 2013.

# Chapter 6

## Results summary

In this section, a general overview of the experimental results and a collection of all the thoughts and ideas that were in all the previous chapters are presented here. Most of the experiments were performed using seven datasets of hyperspectral data that are listed in Table 6.1. We carried out experiments to test each of the distinct methods that make use of compact data structures to attain better performance in terms of storage size and access time. The results are analyzed to determine if these methods can provide a better compression ratio and random element access when compared with other compression techniques.

Note that to evaluate access time, the `getCell` function in the original paper of  $k^2$ -raster is implemented [6]. The storage size of compressed hyperspectral data is measured as bits per pixel per band (bpppb).

### 6.1 Use of different $k$ -values

From experiments in Chapter 3, we found that if different  $k$ -values were used for  $k^2$ -raster, different bit rates and different access times would be produced. The sizes for  $L_{max}$  and  $L_{min}$  are usually not the same for each  $k$ -value, and therefore, the overall size of the structure will not be the same either. In Chapter 3, a heuristic method has been designed to determine the best  $k$ -value for obtaining the smallest storage size. We calculate the sizes ( $s \times s$ ) of the extended matrix for different  $k$ -values within a

Table 6.1: Testing datasets used. In the ‘‘Original Sizes’’ column,  $x$  is the image width,  $y$  the image height and  $z$  the number of spectral bands.

Sensor	Name	Data type	Acronym	Original Sizes ( $x \times y \times z$ )	Bit depth	Best $k$ -value
AIRS	9	Uncalibrated	AG9	$90 \times 135 \times 1501$	12	6
	16	Uncalibrated	AG16	$90 \times 135 \times 1501$	12	6
	60	Uncalibrated	AG60	$90 \times 135 \times 1501$	12	6
	126	Uncalibrated	AG126	$90 \times 135 \times 1501$	12	6
	129	Uncalibrated	AG129	$90 \times 135 \times 1501$	12	6
	151	Uncalibrated	AG151	$90 \times 135 \times 1501$	12	6
	182	Uncalibrated	AG182	$90 \times 135 \times 1501$	12	6
	193	Uncalibrated	AG193	$90 \times 135 \times 1501$	12	6
AVIRIS	Yellowstone sc. 00	Calibrated	AC00	$677 \times 512 \times 224$	16	6
	Yellowstone sc. 03	Calibrated	AC03	$677 \times 512 \times 224$	16	6
	Yellowstone sc. 10	Calibrated	AC10	$677 \times 512 \times 224$	16	6
	Yellowstone sc. 11	Calibrated	AC11	$677 \times 512 \times 224$	16	6
	Yellowstone sc. 18	Calibrated	AC18	$677 \times 512 \times 224$	16	6
	Yellowstone sc. 00	Uncalibrated	AU00	$680 \times 512 \times 224$	16	9
	Yellowstone sc. 03	Uncalibrated	AU03	$680 \times 512 \times 224$	16	9
	Yellowstone sc. 10	Uncalibrated	AU10	$680 \times 512 \times 224$	16	9
	Yellowstone sc. 11	Uncalibrated	AU11	$680 \times 512 \times 224$	16	9
	Yellowstone sc. 18	Uncalibrated	AU18	$680 \times 512 \times 224$	16	9
CRISM	frt000065e6_07_sc164	Uncalibrated	CR1	$640 \times 420 \times 545$	12	6
	frt00008849_07_sc165	Uncalibrated	CR2	$640 \times 450 \times 545$	12	6
	frt0001077d_07_sc166	Uncalibrated	CR3	$640 \times 480 \times 545$	12	6
	hrl00004f38_07_sc181	Uncalibrated	CR4	$320 \times 420 \times 545$	12	5
	hrl0000648f_07_sc182	Uncalibrated	CR5	$320 \times 450 \times 545$	12	5
	hrl0000ba9c_07_sc183	Uncalibrated	CR6	$320 \times 480 \times 545$	12	5
Hyperion	Agricultural	Calibrated	HC1	$256 \times 3129 \times 242$	12	8
	Coral Reef	Calibrated	HC2	$256 \times 3127 \times 242$	12	8
	Urban	Calibrated	HC3	$256 \times 2905 \times 242$	12	8
	Erta Ale	Uncalibrated	HU1	$256 \times 3187 \times 242$	12	8
	Lake Montana	Uncalibrated	HU2	$256 \times 3176 \times 242$	12	8
	Mt. St. Helena	Uncalibrated	HU3	$256 \times 3242 \times 242$	12	8
IASI	Level 0 1	Uncalibrated	IASI1	$60 \times 1528 \times 8359$	12	12
	Level 0 2	Uncalibrated	IASI2	$60 \times 1528 \times 8359$	12	12
	Level 0 3	Uncalibrated	IASI3	$60 \times 1528 \times 8359$	12	12
	Level 0 4	Uncalibrated	IASI4	$60 \times 1528 \times 8359$	12	12

suitable range using the following equation:

$$s = k^{\lceil \log_k \max(w, h) \rceil}, \quad (6.1)$$

where  $w$  is the width and  $h$  the height of the original matrix.

Then, we find the  $k$ -value that corresponds to the matrix with the smallest size,

and the result can be considered as the best  $k$ -value.

### 6.1.1 Storage size

Our testing suggests that in order to get the smallest storage size, we should look for a  $k$ -value that produces the smallest or almost smallest extended matrix size. Note that, in a few cases, when  $k = 2$ , the compressed data size turns out to be larger, or sometimes it is even larger than the original data. But starting from  $k = 3$ , the compressed data size usually gets smaller as the  $k$ -value becomes larger.

By examining Table 9 in Chapter 3, we can choose the best  $k$ -value computed by the heuristic method. However, when the testing data are run through a range of  $k$ -values, the one that produces the smallest storage size is not necessarily the same as that shown in Table 9. For example, for ACY00 (AVIRIS Yellowstone 00), the storage size was found to be the smallest when  $k = 6$ . However, the heuristic approach gives the best storage size when  $k = 3$  or  $k = 9$ . In this case, if we use  $k = 9$  instead of  $k = 6$ , the compressed size is only slightly larger ( $9.69 - 9.61 = 0.08$  bpppb), and the difference is rather insignificant. Hence, the suggested  $k$ -value can be used.

### 6.1.2 Access time

When it comes to access time, for almost all of our hyperspectral data, access time gets smaller when the  $k$ -value becomes larger, and this can be seen in Figure 6 of Chapter 3. There is a trade-off between access time and size with respect to the  $k$ -value. In the case where we are looking for an optimal  $k$ -value, we might select one that produces a comparatively small storage size but with a minimal access time. For example, in AIRS Granule 9, although there is not much difference in storage size for most  $k$ -values, there is a larger decrease in access time when the  $k$ -value becomes larger. Therefore, for AIRS Granule 9, we can consider  $k = 15$  as an optimal value. Note that the optimal value that we are discussing in this section is different from the best  $k$ -value in the previous subsection, as it takes into account the storage size only.

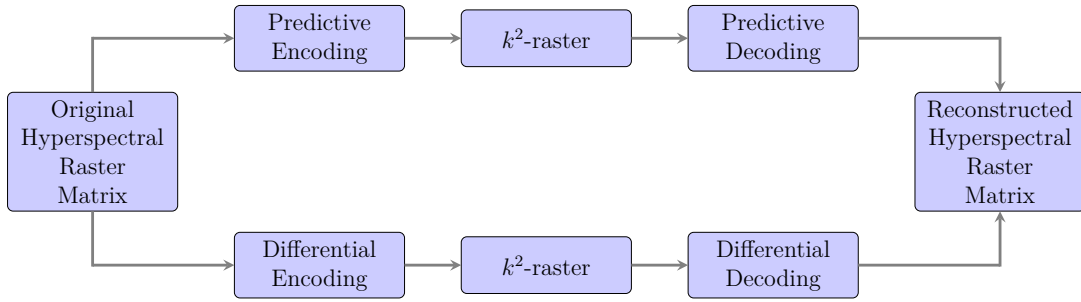


Figure 6.1: A chart showing how the predictive and differential encoders are used with  $k^2$ -raster.

## 6.2 Predictive and differential methods

To take advantage of the spectral redundancies between corresponding elements in neighboring bands in a hyperspectral image, a predictor or a differential encoder can be designed and used to help reduce the bit rate. The resulting data bit rate can be further reduced by using a compact data structure such as  $k^2$ -raster. Figure 6.1 shows the interconnection between the predictive and differential encoders, and  $k^2$ -raster.

The predictive encoder in this thesis is based on three similar compression schemes, the first being Context Adaptive Lossless Image Compression (CALIC) [21] published in 1994, the second 3D-CALIC [22] in 2000 and the third M-CALIC [23] in 2004. The first two were proposed by Wu et al. and the last one by Magli et al. The predictive method takes advantage of the fact that neighboring pixels in the same band (spatial correlation) typically have similar data values and the values of the corresponding elements in neighboring bands can be even closer (spectral correlation).

On the other hand, the differential encoder, being a special case of the predictive encoder, only takes the difference between the corresponding elements in neighboring bands. Therefore, it only takes advantage of the spectral redundancies. It should be noted that the correlation between corresponding pixels in the bands become higher as the bands are closer to each other.

### 6.2.1 Storage size

Experiments were carried out to compare the performance of  $k^2$ -raster with other popularly used algorithms such as Gzip, bzip2, and xz, and with specialized algorithms such as M-CALIC and CCSDS 123.0-B-1. These experiments were repeated for  $k^2$ -raster with the predictor and the differential encoder. The results show that using  $k^2$ -raster by itself already performs better than Gzip.

With the predictor added, it produces similar or better results when compared with other compression techniques such as bzip2 and xz. We should note that the differential encoder is a special case of the predictor and does not always give optimal results. Therefore, the predictor is generally expected to perform better than the differential encoder.

The results also indicate that the performance of CCSDS 123.0-B-1 and M-CALIC is better than that of the others as these two schemes are compression techniques that are rather specialized, and, in particular, CCSDS 123.0-B-1 is used as a baseline against which all compression algorithms for hyperspectral images are compared.

### 6.2.2 Access time

As for access time, it takes longer to access a cell in a  $k^2$ -raster structure with a predictor than without one. This is to be expected although we need to realize that the advantage of a predictor is that it reduces the bit rates which make the storage size smaller and the transmission faster. Also, if we look at Table 4 in Chapter 2, the Gzip decompression times generally take at least four or five times more than when a predictor is used to randomly access the data using the `getCell` function.

The predictive and differential methods are compared to Reversible Haar Transform at levels 1 and 5. The predictive method performs better than others except for Reversible Haar Transform level 5. The reason is that for reverse transformation, only two pixels (reference and current pixels) are needed for the predictive and differential methods while Reversible Haar Transform at level 5 requires a more complicated process for decoding.

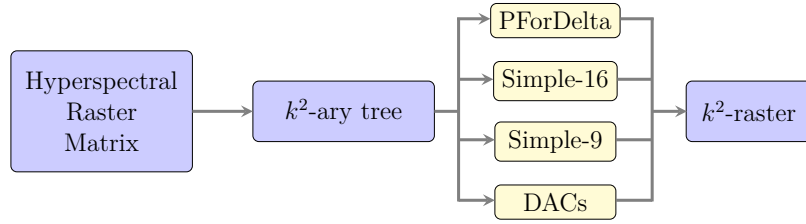


Figure 6.2: Construction of  $k^2$ -raster. A  $k^2$ -ary tree is first built from a raster matrix. Compression and random access are achieved when tree node data are encoded by an integer encoder, such as DACs, Simple-9, Simple-16, or PForDelta, resulting in a  $k^2$ -raster structure.

### 6.2.3 Group size

The predictor and the differential encoder that use different group sizes produce different bit rates. For our testing, different group sizes were used: 2, 4 and multiples of 4 up to 32. We found that for most test data, the bit rates are at their optimum when the group size is either 4 or 8. However, we should note that the predictor always produces lower bit rates than the differential encoder does, regardless of what the group size is. From the experiments, we suggest that a group size of 4 should be used for most general cases.

## 6.3 Integer encoders

For compression and random access to work well in a compact data structure such as  $k^2$ -raster, a variable-length encoder or integer encoder is needed. Some of them include DACs, Rice, Simple-9, Simple-16, and PForDelta codes. Figure 6.2 shows how the  $k^2$ -raster is built with different integer encoders from a raster matrix.

Experiments that were performed for integer encoders can be found in Chapters 3 and 4.

### 6.3.1 Storage size

In Table I of Chapter 4, under the “Padded Matrix” column, we can see that PForDelta using 128-integer blocks has the best results for most of the test data. DACs works

best with CRSIM data and Simple-16 with IASI data.

### 6.3.2 Access time

Similar to storage size, experiments for hyperspectral data show that PForDelta performs better than DACs with respect to random access. In Table II of Chapter 4, under the “Padded Matrix” column, the best access time is to use the 32-, 64-, and 128-integer PForDelta encoders.

## 6.4 Padding versus unpadding

Given a  $k$ -value, if the dimensions (width ( $w$ )  $\times$  height ( $h$ )) of the original raster matrix are not a power of  $k$ , then this matrix needs to be extended to size  $s \times s$ , where  $s$  is computed according to the Equation 6.1 in Section 6.1.

All the new cells will be set to zero, and the new matrix is called a padded matrix. The reason is because to arrive at a particular location starting from the root down to its leaves, the search requires that we find its child node location by using the rank function and if the number of child nodes is not equal to  $k^2$ , the search path will not be correct. Adding new cells also means the nodes that are outside the original matrix have to be saved, which may lead to a larger structure. To reduce the storing of redundant data, we propose using an unpadded matrix which means only nodes that are inside the original matrix will be saved.

Experiments that were performed for padded and unpadded matrices can be found in Chapter 4.

### 6.4.1 Storage size

From our experiments, it can be shown that there are up to 6% in savings if unpadded matrices are used instead of padded ones. For most padded and unpadded matrices, PForDelta-128 produces the best results followed by DACs and Simple-16. Overall, the storage size has improved if an unpadded matrix is used with one of the integer encoders, especially PForDelta.



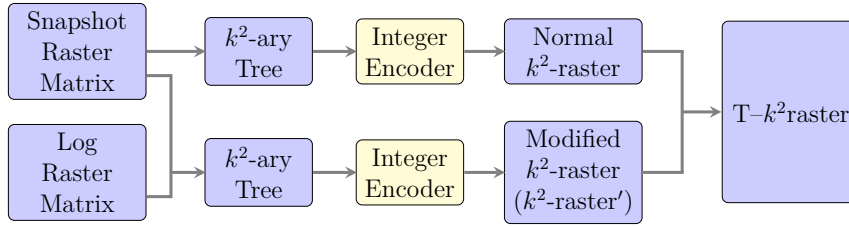


Figure 6.3: Construction of T- $k^2$ raster. A  $k^2$ -ary tree is first built from a snapshot raster matrix and/or log raster matrix. Compression and random access are achieved when tree node data are encoded by an integer encoder such as DACs, Simple-16, or PForDelta, producing a  $k^2$ -raster or a  $k^2$ -raster' structure. Finally, these two structures are combined to form a T- $k^2$ raster structure.

### 6.4.2 Access time

To optimize random cell access in an unpadded matrix, access times are made faster when the `getCell` function are modified with partial sums and sampling functions [1, §3.3 and §4.2]. However, we should caution that the use of these functions might introduce overhead and more memory usage. Access times are reduced almost in half for most of the unpadded matrices. The exceptions are AIRS and IASI data which have more or less the same access times for both kinds of matrices. This might be the result of padded and unpadded matrices having similar tree data. Overall, for padded matrices, PForDelta is most definitely the winner for best access time while for unpadded matrices, PForDelta shares it with DACs.

## 6.5 T- $k^2$ raster structure

T- $k^2$ raster is a recently proposed compact data structure that was designed to exploit the temporal regularities between consecutive rasters in a time series [7]. Raster time series or temporal rasters are raster images that cover the same region at consecutive timestamps. The research group that proposed this structure has produced some favorable results for raster time series data. In this thesis, we study the use of this structure for hyperspectral data. The T- $k^2$ raster structure is built by regularly grouping the raster matrices that contain a combination of snapshots and logs, where the group size is denoted by  $t_\delta$  and  $t_\delta \geq 2$ . In other words, the consecutive bands

are divided into groups of two or more. In each group, the first raster ( $M_s$ ) is the snapshot raster and subsequent rasters ( $M_{s+i}$ ) are the log rasters. Elements in these log rasters ( $M_{s+i}$ ) are compared to the corresponding elements in the first raster ( $M_s$ ) and the differences are taken, and as a result a modified  $k^2$ -raster' is created.

To optimize this structure, in each group, we will build the first band as a  $k^2$ -raster while the others could be built as either a  $k^2$ -raster or  $k^2$ -raster', depending on the size of the structure that is produced. This means that it takes advantage of the spatial redundancies or the spectral redundancies, whichever resulting in a better storage size. The grouping of bands is still considered as regular. Figure 6.3 shows how to construct a T- $k^2$ raster.

The heuristic version T<sub>H</sub>- $k^2$ raster goes even one better than the above optimized version. It analyzes the bands starting from the first one and decides whether the band that follows should be built as a  $k^2$ -raster or  $k^2$ -raster', depending on the size produced. As we are moving along down the bands, if one band is built as a  $k^2$ -raster' and its previous band is a  $k^2$ -raster', we could transform this previous band into a  $k^2$ -raster and the current band could be rebuilt as a  $k^2$ -raster' with respect to the new previous  $k^2$ -raster band. The choice is made by evaluating whether the old or the new combination can provide better improvement. Therefore, T<sub>H</sub>- $k^2$ raster again strives to get the best compression by choosing between spatial correlation and spectral correlation for building the bands, and it may change the previous band whenever it is deemed appropriate. As the results show, T<sub>H</sub>- $k^2$ raster has the best overall storage sizes for all our datasets. The grouping of bands in this heuristic version is irregular.

### 6.5.1 Storage size

T<sub>H</sub>- $k^2$ raster gives the best results for most datasets when it is used with DACs or PForDelta-128, while the IASI dataset performs best with Simple-16. T<sub>H</sub>- $k^2$ raster is shown to reduce our hyperspectral test data up to 19% compared to  $k^2$ -raster or up to 61% compared to the uncompressed data. For storage size, due to the nature of T<sub>H</sub>- $k^2$ raster, it always performs better than T- $k^2$ raster, which in turn, performs better than  $k^2$ -raster alone.

### 6.5.2 Access time

For access time, with  $T_H$ - $k^2$ -raster, there is some improvement in access time to data elements even though the process to access them is actually becoming a bit more complicated. This can be explained by the fact that the overall structure in  $T_H$ - $k^2$ -raster has become smaller, thereby making it faster to query elements in the DACs-compressed data. For IASI data, results for  $T_H$ - $k^2$ -raster are only  $0.1 \mu s$  slower than  $k^2$ -raster.

# Chapter 7

## Conclusion

Since the dawn of computers, a huge amount of digital data have been produced by individual users, the business world, and users in other domains such as government, science, health care, engineering etc., and they are accumulating and growing at an exponential rate. It is estimated that the compound growth rate of these data is currently at 23% and by 2025 the amount of data will possibly reach 181 zettabytes (1 zettabyte =  $10^{21}$  bytes). Therefore, their transmission and storage are some of the challenges that we are now facing. It is incumbent upon us to find some ways to tackle them, and one of the solutions to save space is to use data compression.

Like any other digital data, hyperspectral data, with their large sizes, need to be compressed for faster transmission and smaller storage size. As mentioned earlier, these data are useful for extracting information to be used in diverse applications such as weather prediction, wildfire soil studies, oil field exploration and even tea leaf cultivation. In this thesis, compact data structures such as  $k^2$ -raster and T- $k^2$ -raster have been used to compress these data, with the additional benefit of random data access without having to be fully decompressed. The experimental results that we have conducted have shown that they are competitive when it comes to performances such as storage sizes and random cell access.

## 7.1 Summary

This section summarizes the research that has been carried out in this thesis and the contributions that have been made in the field of compact data structures. Additionally, a few thoughts on things that I have learned and discovered along the way are included.

- In Chapter 2, it can be seen that  $k^2$ -raster can reduce the bit rates of hyperspectral data. If the data is applied with the predictive or differential method, the rates can be further reduced significantly. The predictor produces a better reduction in bit rates than the differential encoder and is preferred to be used for hyperspectral data. Although the access time turns out to be a little slower with these methods, we should understand that the elements in the raster matrix can be accessed without having to go through any kind of decompression and that is a big plus for using  $k^2$ -raster alone or using  $k^2$ -raster with either predictive method or differential method. For decompressing data done by other compression techniques such as Gzip, the process would take 4 or 5 times much longer.
- In Chapters 3 and 4, several variable-length encoders or integer encoders have been tested in this thesis such as Rice codes, the Simple family and PForDelta and have been found to perform favorably compared with DACs. We should add that the DACs version that we have used in our experiments are optimized. For the Simple family codes, we try to optimize them by adding partial sums and sampling functions in our implementations. When sampling is used, the search on the compressed array with  $n$  elements has changed from  $\mathcal{O}(n)$  to  $\mathcal{O}(1)$  which means the time performance changes from linear to constant. Also, this gives us some new ideas of designing codes with integrated sampling function in future projects, and this will be explained further in the next subsection “Future work”.
- In Chapter 4, experiments using unpadded matrices turned out to be quite competitive compared with padded matrices. Similar to integer encoders, sampling

codes must be added to the element access code for it to work well and well enough to even outperform that of DACs. As mentioned previously, this adds memory overhead and users need to take this into account when using unpadded matrices.

- In Chapter 5, we have examined the recently proposed T- $k^2$ raster, which is a combination of  $k^2$ -raster and a modified form of  $k^2$ -raster (or  $k^2$ -raster'). There are two versions for  $k^2$ -raster, one being a normal version, where the bands are regularly grouped, and the other being a heuristic one, where the bands are irregularly grouped. The experimental results with T- $k^2$ raster and T<sub>H</sub>- $k^2$ raster in Chapter 5 have shown that both structures can improve the compression ratio when, building the structure, each spectral band is given a choice of taking the spatial differences or the spectral differences. The one that provides a better compression ratio will be used. T<sub>H</sub>- $k^2$ raster is simply a more optimized version of T- $k^2$ raster. Hence, T<sub>H</sub>- $k^2$ raster will always perform better than T- $k^2$ raster, which, in turn, performs better than  $k^2$ -raster, a structure that only exploits spatial redundancies. As for random access, T- $k^2$ raster also performs better than  $k^2$ -raster. Previously T- $k^2$ raster worked well with raster time series, and in this thesis, this structure, especially the heuristic version, has proven to work equally well with hyperspectral data. It is fair to say that the structure can be generalized to work with 3D raster data and should produce favorable results compared to  $k^2$ -raster.
- The software in this thesis was written in C and C++. They are compiled programs and generally produce faster executables. The programs can be found on our department website: <https://gici.uab.cat/GiciWebPage/downloads.php>.

## 7.2 Future work

In this section, we describe a number of plans or projects that we think should be interesting to pursue in the future:

- We can assess the possibility of modifying elements in a  $k^2$ -raster structure for hyperspectral data. Up to this moment, only randomly reading or querying of data from the structure is possible. But modifying or replacing elements in the structure is still an area that is largely unexplored, and it will be worthwhile looking into. If this can be achieved, then we can replace elements without the need to decompress and compress the whole structure. This research line is based on the discussion on using a dynamic structure called  $dk^2$ -tree, proposed by de Bernardo et al. [24, 25], and on a chapter entitled “Dynamic  $k^2$ -Trees” from the book “Compact Data Structures” by Navarro [1, §12.5.2].
- There were a number of compact data structures that we would have liked to work on from the outset of our research. But due to time constraints, we have only been able to focus on  $k^2$ -raster and its variants, and the favorable results that we have been getting has motivated us to keep on going in this direction of research. However, we can go in a different direction now and explore other compact data structures to be used for hyperspectral data. One of them is called balanced sequence of parentheses can be used to build trees in a more succinct manner so that the resulting structure can be more compact. This idea is based on a paper “Fully Functional Static and Dynamic Succinct Trees” by Navarro and Sadakane [26], and on a chapter on this data structure entitled “Parenthesis” from the book “Compact Data Structures” by Navarro [1, §7]. Other compact data structures that are of interest are wavelet trees and dynamic structures, to name a few.
- For  $T-k^2$ -raster, we can consider reordering the rasters in the different bands of the hyperspectral data to achieve better performance. Additionally,  $T_H-k^2$ -raster can be treated as an optimization problem and better heuristics can be found to improve the compression ratio.
- The Simple-9, Simple-16 and PForDelta encoders that have been studied in this

thesis do not have any built-in sampling functions. It would be interesting to design an encoder with the same functionalities but also with intrinsic sampling capabilities. This would help reduce the additional memory allocated for sampling when queries are performed.





# Bibliography

- [1] G. Navarro, *Compact data structures: A practical approach*. Cambridge University Press, 2016.
- [2] G. J. Jacobson, “Succinct static data structures,” Ph.D. dissertation, Carnegie Mellon University, 1988.
- [3] G. Jacobson, “Space-efficient static trees and graphs,” in *30th Annual Symposium on Foundations of Computer Science*. IEEE, 1989, pp. 549–554.
- [4] *Low-Complexity Lossless and Near-Lossless Multispectral and Hyperspectral Image Compression. Blue Book. Issue 2*, Consultative Committee for Space Data Systems (CCSDS) Std. CCSDS 123.0-B-2, Feb. 2019. [Online]. Available: <https://public.ccsds.org/Pubs/123x0b2c3.pdf>
- [5] N. R. Brisaboa, S. Ladra, and G. Navarro, “ $k^2$ -trees for compact web graph representation,” in *International Symposium on String Processing and Information Retrieval*. Springer, 2009, pp. 18–30.
- [6] S. Ladra, J. R. Paramá, and F. Silva-Coira, “Scalable and queryable compressed storage structure for raster data,” *Information Systems*, vol. 72, pp. 179–204, December 2017.
- [7] F. Silva-Coira, J. R. Parama, G. de Bernardo, and D. Seco, “Space-efficient representations of raster time series,” *Information Sciences*, vol. 566, pp. 300–325, 2021.

- [8] F. F. Sabins, "Remote sensing for mineral exploration," *Ore geology reviews*, vol. 14, no. 3-4, pp. 157–183, 1999.
- [9] R. D. M. Scafutto, C. R. de Souza Filho, and W. J. de Oliveira, "Hyperspectral remote sensing detection of petroleum hydrocarbons in mixtures with mineral substrates: Implications for onshore exploration and monitoring," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 128, pp. 146–157, 2017.
- [10] Y. Huang, W. Dong, A. Sanaeifar, X. Wang, W. Luo, B. Zhan, X. Liu, R. Li, H. Zhang, and X. Li, "Development of simple identification models for four main catechins and caffeine in fresh green tea leaf based on visible and near-infrared spectroscopy," *Computers and Electronics in Agriculture*, vol. 173, p. 105388, 2020.
- [11] P. R. Robichaud, S. A. Lewis, D. Y. Laes, A. T. Hudak, R. F. Kokaly, and J. A. Zamudio, "Postfire soil burn severity mapping with hyperspectral image unmixing," *Remote Sensing of Environment*, vol. 108, no. 4, pp. 467–480, 2007.
- [12] J. Le Marshall, J. Jung, T. Zapotocny, J. Derber, R. Treadon, S. Lord, M. Goldberg, and W. Wolf, "The application of AIRS radiances in numerical weather prediction," *Australian Meteorological Magazine*, vol. 55, no. 3, pp. 213–217, 2006.
- [13] N. R. Brisaboa, S. Ladra, and G. Navarro, "DACs: Bringing direct access to variable-length codes," *Information processing & management*, vol. 49, no. 1, pp. 392–404, 2013.
- [14] R. Rice and J. Plaunt, "Adaptive variable-length coding for efficient compression of spacecraft television data," *IEEE Transactions on Communication Technology*, vol. 19, no. 6, pp. 889–897, 1971.
- [15] V. N. Anh and A. Moffat, "Inverted index compression using word-aligned binary codes," *Information Retrieval*, vol. 8, no. 1, pp. 151–166, 2005.

- [16] J. Zhang, X. Long, and T. Suel, “Performance of compressed inverted list caching in search engines,” in *Proceedings of the 17th international conference on World Wide Web*, 2008, pp. 387–396.
- [17] M. Zukowski, S. Heman, N. Nes, and P. Boncz, “Super-scalar RAM-CPU cache compression,” in *22nd International Conference on Data Engineering (ICDE’06)*. IEEE, 2006, pp. 59–59.
- [18] K. Chow, D. E. O. Tzamarias, I. Blanes, and J. Serra-Sagristà, “Using predictive and differential methods with  $k^2$ -raster compact data structure for hyperspectral image lossless compression,” *Remote Sensing*, vol. 11, no. 21, 2019. [Online]. Available: <http://dx.doi.org/10.3390/rs11212461>
- [19] K. Chow, D. E. O. Tzamarias, M. Hernández-Cabronero, I. Blanes, and J. Serra-Sagristà, “Analysis of variable-length codes for integer encoding in hyperspectral data compression with the  $k^2$ -raster compact data structure,” *Remote Sensing*, vol. 12, no. 12, 2020. [Online]. Available: <http://dx.doi.org/10.3390/rs12121983>
- [20] K. Chow, D. E. O. Tzarmarias, M. Hernández-Cabronero, I. Blanes, and J. Serra-Sagristà, “Performance improvement on  $k^2$ -raster compact data structure for hyperspectral scenes,” *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2022. [Online]. Available: <http://dx.doi.org/10.1109/LGRS.2021.3084065>
- [21] X. Wu and N. Memon, “CALIC - a context based adaptive lossless image CODEC,” in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, 1996.
- [22] ———, “Context-based lossless interband compression - extending CALIC,” *IEEE Transactions on Image Processing*, vol. 9, no. 6, pp. 994–1001, 2000.
- [23] E. Magli, G. Olmo, and E. Quacchio, “Optimized onboard lossless and near-lossless compression of hyperspectral data using CALIC,” *IEEE Geoscience and remote sensing letters*, vol. 1, no. 1, pp. 21–25, 2004.

- [24] G. de Bernardo, S. Álvarez García, N. R. Brisaboa, G. Navarro, and O. Pedreira, “Compact querieable representations of raster data,” in *International Symposium on String Processing and Information Retrieval*. Springer, Cham, October 2013, pp. 96–108.
- [25] N. R. Brisaboa, G. de Bernardo, and G. Navarro, “Compressed dynamic binary relations,” in *2012 Data Compression Conference*. IEEE, April 2012, pp. 52–61.
- [26] G. Navarro and K. Sadakane, “Fully functional static and dynamic succinct trees,” *ACM Transactions on Algorithms (TALG)*, vol. 10, no. 3, pp. 1–39, 2014.