# Compact Querieable Representations of Raster Data*

Guillermo de Bernardo[1], Sandra Álvarez-García[1], Nieves R. Brisaboa[1], Gonzalo Navarro[2], and Oscar Pedreira[1]

[1] Databases Lab., University of A Coruña, Spain
[2] Department of Computer Science, University of Chile, Chile

**Abstract.** In Geographic Information Systems (GIS) the attributes of the space (altitude, temperature, etc.) are usually represented using a raster model. There are no compact representations of raster data that provide efficient query capabilities. In this paper we propose compact representations to efficiently store and query raster datasets in main memory. We experimentally compare our proposals with traditional storage mechanisms for raster data, showing that our structures obtain competitive space performance while efficiently answering range queries involving the values stored in the raster.

## 1 Introduction

The raster model is widely used to represent spatial attributes [14]. A raster is essentially a matrix representing a region of the space, in which the space is split into cells and a value of the spatial attribute is stored for each of these cells. An uncompressed raster representation requires much space (e.g., a $50,000 \times 50,000$ grid of integers requires around 10 GB), so it is typically stored in secondary memory. Compressed raster representations are mainly designed to reduce storage, and are based on well-known compression techniques such as run-length encoding or LZW [15]. In these representations the full file must be decompressed even to display a small region of the space. Some representations split the raster into fixed size tiles and compress each tile independently, providing some level of direct access to regions and taking advantage of the locality of values to enhance compression (for example, $GeoTIFF$[3] images can be used to represent raster data and they support this partition into tiles with different compression techniques including LZW).

Geographic Information Systems (GIS) [16, 14] routinely make use of raster data to represent various kinds of information. They usually need not only direct access to regions (e.g. to display a local map), but also need to *find* the cells whose value is within some range. A classic example is the visualization of pressure or

* GdB, NB, SAG and OP were funded by MICINN (PGE and FEDER) grants TIN2009-14560-C03-02, TIN2010-21246-C02-01 and CDTI CEN-20091048, and by Xunta de Galicia (co-funded with FEDER) ref. 2010/17. GN was founded by Millennium Nucleus Information and Coordination in Networks ICM/FIC P10-024F.

[3] http://trac.osgeo.org/geotiff/

temperature bands, which require retrieving the coordinates that have values within a given range. Another example is retrieving the regions of a raster with an altitude above a given threshold to find zones with snow alert, or below a value to find regions with risk of floods. However, usual raster representations lack *indexing* capabilities on the values stored in the raster. These representations need to traverse the complete raster in order to return the cells that contain a given value, even when the results may be restricted to a small subregion of the space.

One solution is to consider the raster as a 3-dimensional matrix and use computational geometry solutions to answer all these queries as range reporting queries [5]. However, these solutions require superlinear space and therefore they are not suitable to the large datasets involved. Reading the raster row-wise and storing the sequence of values we could use a compressed sequence representation [10, 9, 1] to return the cells with a given value (or a range of values [10, 13]) efficiently, but further restricting the search to a spatial range is not efficiently handled. Furthermore, these sequence representations achieve at best the zero-order entropy space of the sequence, and this is not a significant space reduction in many cases.

In this paper we present several proposals that aim at providing at the same time a compact representation of raster data and efficient support of queries involving spatial windows and intervals of values. We design our structures to solve queries such as retrieving all the values of a given area, retrieving all the coordinates with a given value, or retrieving all the entries of the raster within a spatial window and with values in a given range. Our structures are enhancements of an existing data structure called k$^2$-tree [4], originally designed to represent sparse binary matrices. Our first contribution is a variant of the k$^2$-tree that can compress not only large regions of zeros but also regions of ones. This enhancement allows our structure to compress efficiently not only sparse matrices but also binary images that contain large homogeneous regions. We experimentally compare our structure with a Linear Quadtree [8] representation showing its superiority in space and even time. Our second contribution is a generalization of the k$^2$-tree to represent multi-dimensional data. We call this structure a k$^n$-tree. We use these new structures to provide different representations of raster data, each with different strengths. We test our proposals experimentally to demonstrate their low space requirements and their ability to efficiently solve queries. Finally, we describe other application domains where our proposals could be of interest.

## 2 Previous Work: The k$^2$-tree

The k$^2$-tree [4] is a data structure for the compact representation of sparse binary matrices. In this paper we use its simplest variant, k=2, so the k$^2$-tree is similar to a compact Quadtree [7]. It corresponds to a recursive partition of the binary matrix. At each partitioning step, the matrix is divided into $k^2$ submatrices of equal size. Each submatrix is represented using a single bit: 1 if the submatrix

contains at least one 1, or 0 otherwise. The method proceeds recursively for each 1-child until the current submatrix is full of 0s or we reach the cells of the original matrix. This conceptual tree is traversed levelwise and stored in two bit arrays: $T$ stores all the levels except the last one, and $L$ stores the last level. Figure 1 shows an example of k$^2$-tree. In order to navigate the tree we need to build a *rank structure* over T. This structure stores a set of counters that allow us to compute the number of ones in the bitmap up to any position ($rank_1$ operation) in constant time using sublinear space [12]. Given a value 1 at position $pos$ in $T$, its $k^2$ children will start at position $pos' = rank_1(T, pos) \times k^2$ of $T : L$. This property provides simple navigation over the conceptual tree using only the bitmaps and the additional rank structure over $T$. A k$^2$-tree can solve single cell queries, row/column queries or general range reporting queries (i.e., report all the 1s in a range) using only rank operations, by visiting all the necessary subtrees.

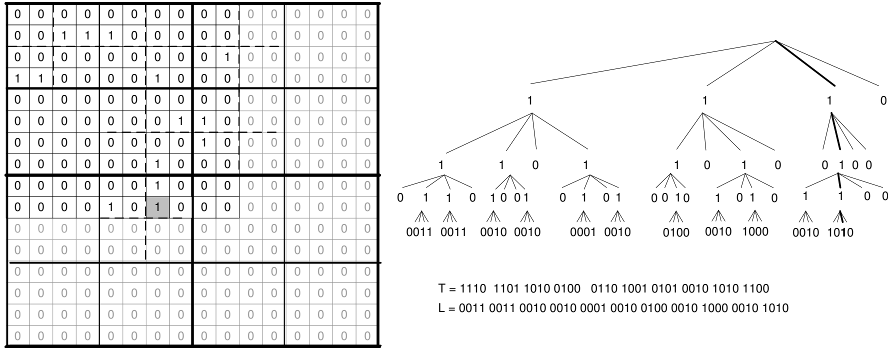


**Fig. 1.** Binary matrix and its k$^2$-tree representation, for k=2. The matrix is virtually expanded to the next power of k.

The original k$^2$-tree was designed as a static data structure. A dynamic variant of the k$^2$-tree, called dk$^2$-tree, also exists [3]. The dk$^2$-tree essentially splits the bitmaps $T$ and $L$ in chunks and builds tree structures to store these chunks. The internal nodes of the trees store counters that replace those of static rank structures and allow access to specific positions in $T$ or $L$. The dk$^2$-tree provides the same query capabilities of the original k$^2$-tree but allows at the same time update operations, including changes in the values of the cells and insertion of new rows/columns at the end of the matrix. See the original paper [3] for further details.

## 3 Compression of Ones

In this section we propose variants of the k$^2$-tree that are able to compress efficiently these large regions of ones and zeros. We will show 2 variants: the first performs better when the number of ones and zeros in the matrix is not too

different; the second variant is designed to be used when the proportion of zeros and ones is very different (without loss of generality, we will consider that the less frequent value is 1).

The idea behind our proposals is to stop the decomposition of the binary matrix when a uniform region is found, be it of zeros or ones. This means that in our $k^2$-tree we must discriminate among the 3 possible "colors" of a node: white nodes are regions of zeros, black nodes are regions of ones and gray nodes are internal nodes that correspond to regions with zeros and ones. With this information, large regions of ones can be represented using a single node, just like regions of zeros in the original $k^2$-tree. The variants with compression of ones can be traversed like the original $k^2$-trees, and provide in addition a way to find the color of a node. Using the additional navigation rules, all the operations supported by original $k^2$-trees can be implemented directly in a $k^2$-tree with compression of ones. The algorithms must only be adapted to expand automatically all the results that fall in the submatrix covered by a black node. We now propose two specific representations of the color of nodes.

### 3.1 2-bits Variant

In this variant, we mark with a 1 all the regions that contain both zeros and ones (i.e., the internal or gray nodes), whereas uniform regions are assigned a 0. In order to tell apart white from black nodes, we create a second bitmap $T'$ in addition to $T$, that stores the value of each uniform region (that is, $T'$ contains a bit for each 0 in $T$ that will store the color of that region). The navigational properties of the original $k^2$-tree still hold: the children of the (gray) node at position $p$ will start at position $p' = rank_1(T, p) \times k^2$, because each bit set to 1 in $T$ represents a gray node and only gray nodes have children. If a position $p$ in $T$ is set to 0, we can check $T'[rank_0(T, p)]$ to see if it corresponds to a region of zeros or of ones. The bitmap $L$ behaves as in original $k^2$-trees.

Figure 2 shows a $k^2$-tree with compression of ones and the bitmaps generated for this variant (left). We highlight a black node and the positions where its bits are assigned in the bitmaps $T$ and $T'$.

### 3.2 Unbalanced (1-5)-bits Variant

In this variant, we achieve compression of ones using the same bitmaps of the original $k^2$-tree. White nodes will be represented with a 0 and gray nodes will be assigned a 1, exactly like in original $k^2$-trees. Black nodes will be encoded as a gray node with $k^2 = 4$ white children (i.e., they are encoded using 5 bits). This combination, that can not appear in the original $k^2$-tree (gray nodes, by definition, represent regions with at least a 1), is used to represent regions of ones without the need of additional structures. Figure 2 (right) shows the bit distribution for this variant. To take into account regions of ones, at each step of the $k^2$-tree traversal, we must check the $k^2 - 1$ siblings of the current node to detect if the current node is white (the current bit is 0, but one of its siblings is

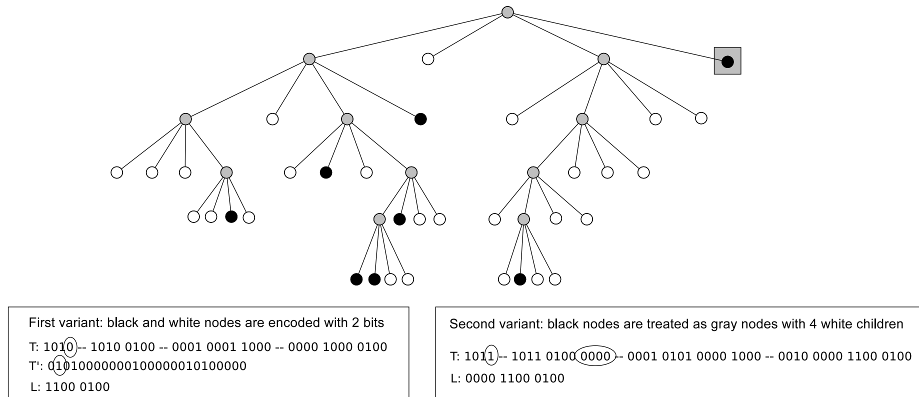


**Fig. 2.** Example of k$^2$-trees with compression of ones.

1)[4] or we are in a region of ones (the current bit is 0 and all its siblings are 0, meaning that the parent node was actually black).

This approach will obtain worse compression than the one based on 2 bits in most cases. However, it may obtain better compression when there are blocks of ones but the zeros are much more frequent. Also, this variant will never use more space than the original k$^2$-tree.

Both variants of k$^2$-trees with compression of ones can also be applied to dynamic k$^2$-trees. Each bitmap used by the static representations is replaced by a tree structure as in the original dk$^2$-tree, to support updates as well as access and rank queries.

### 3.3 Comparison of k$^2$-tree with Linear QuadTree

The QuadTree (QT) [7] is a well-known spatial index structure for representing binary images. The partitioning principle of QT and k$^2$-tree is the same. At the root of the tree, the matrix is partitioned into four quadrants, which correspond to the four children of the root. In the QT, the quadrants that are not entirely black or white are recursively partitioned following the same principle until we reach a fully black or white region or the unitary cells of the matrix. To access a particular position of the raster, the tree is traversed from the root to the leaves following the appropriate path. Depending on the distribution of black and white cells in the matrix, the QT can represent the binary matrix while saving significant space and providing efficient access.

The Linear QuadTree (LQT) [8] was proposed as a representation of QTs without pointers that can be easily managed in secondary memory and requires less space than the original QTs. At each node, the branches corresponding to the NW, NE, SW, and SE quadrants are labelled with 0, 1, 2, and 3 respectively. A leaf is formed when the submatrix is full of points. LQTs assign a *quadcode* to each leaf of the QT, which describes the (4-ary) path from the root to that leaf. To represent the quadcode of each leaf, a digit is added for each branch that is

[4] The $k^2$ bits are contiguous, so we perform this check in constant time

traversed. An additional symbol is used to represent that a region is not further partitioned (i.e., the subregion covered by that branch is full of ones, i.e., it is a leaf). All these quadcodes are then stored in a B-Tree in secondary memory. To access a position of the raster, we search for its corresponding quadcode or a quadcode that contains it in the B-Tree.

Other representations of QTs that achieve less space requirements have been proposed, such as the FBLQ [6] and the CBLQ [11]. However, they are designed mainly to represent binary images and support union, intersection and difference of two images, which require a full traversal of the raster.

The original $k^2$-trees cannot be used to represent a wide class of binary images because they do not efficiently compress regions of ones. Our variants of $k^2$-tree with compression of ones overcome this limitation. As a proof of concept of the capabilities of our proposals we compare our $k^2$-trees with LQTs.

**Experimental comparison** We compare the space requirements and search performance of $k^2$-trees and LQTs. Since $k^2$-trees work in main memory, we implemented in-memory versions of the LQT. To provide fair comparisons, we build two different variants of LQT. The first one (LQT-BTree) stores the quadcodes in a B-Tree maintained in main memory. This representation can handle modifications, so we compare it with a d$k^2$-tree. The second variant (LQT-Array) stores the sorted sequence of quadcodes directly in an array, and a binary search is used to find them. We compare the LQT-Array with a static $k^2$-tree with compression of ones. We run all our experiments on an AMD-Phenom-II X4 955@3.2 GHz, with 8GB DDR2 RAM. The operating system is Ubuntu 9.10. All our implementations are written in C and compiled with gcc version 4.4.1 with -O9 optimizations.

We use five collections in our comparison. The first three are binary images obtained from elevation rasters of the Digital Terrain Model MDT05 of the Spanish Geographic Information Center [5]. A threshold is applied to each raster, generating a binary image with 25% of ones, corresponding to the higher values. The last two collections are adjacency matrices of Web graphs [2], and therefore difficult to compress using LQTs because they are very sparse. Table 1 shows the space required by all the representations. Table 2 shows the average time needed to retrieve a cell of the matrix. To obtain this time, we run a million random queries in each dataset and compute the average access time.

Our results show that the space required by $k^2$-trees is an order of magnitude smaller than the required by our two variants of LQTs, while the access time is also better in most cases, particularly in static $k^2$-trees.

## 4 Multi-dimensional $k^2$-trees: The $k^n$-tree

The $k^2$-tree has been applied in several contexts to the representation of binary relations. Intuitively, the $k^2$-tree can be extended to solve problems of higher

[5] Original rasters are available for download at http://cnig.es/

| Dataset | rows × cols | #ones | k²-tree | | LQT-Array | LQT-BTree |
|---|---|---|---|---|---|---|
| | | | Static | Dynamic | | |
| mdt-600 | 3961×5881 | 11,647,287 | 0.02 | 0.04 | 0.25 | 0.31 |
| mdt-700 | 3841×5841 | 13,732,734 | 0.02 | 0.04 | 0.17 | 0.17 |
| mdt-800 | 3921×6001 | 21,580,638 | 0.01 | 0.02 | 0.11 | 0.11 |
| cnr | 325,557×325,557 | 3,216,152 | 3.14 | 4.95 | 41.32 | 41.46 |
| eu | 862,664×862,664 | 19,235,140 | 3.81 | 5.86 | 49.92 | 50.07 |

**Table 1.** Space utilization of k$^2$-trees and LQTs (in bits per one).

| Dataset | k²-tree | | LQT-Array | LQT-BTree |
|---|---|---|---|---|
| | Static | Dynamic | | |
| mdt-600 | 0.46 | 0.86 | 1.64 | 1.27 |
| mdt-700 | 0.50 | 1.00 | 1.64 | 1.27 |
| mdt-800 | 0.36 | 0.56 | 1.64 | 1.20 |
| cnr | 0.86 | 2.88 | 3.16 | 3.06 |
| eu | 0.90 | 3.85 | 3.76 | 3.85 |

**Table 2.** Time to retrieve the value of a cell of the binary matrix, in $\mu$s/query.

dimensionality extending its space partitioning while maintaining the representation techniques used. We call the extension of the k$^2$-tree a k$^n$-tree.

A k$^n$-tree represents a binary $n$-dimensional matrix $M_{m_1 \times \ldots \times m_n}$ by recursively partitioning it into $k^n$ $n$-dimensional submatrices of equal size. This partitioning strategy generates a conceptual tree similar to a k$^2$-tree in which each node has $k^n$ children. The conceptual tree can then be represented and queried using the same techniques of k$^2$-trees.

Notice that a 1 in a cell of a binary matrix that has another 1 falling in the same submatrix does not consume any additional space in the k$^2$-tree, but isolated ones induce a complete branch in the k$^2$-tree, consuming much space. In other words, the k$^2$-tree takes advantage of the proximity of the ones in the matrix, because paths in the conceptual tree to each of these ones can be shared for most of the levels. This feature becomes more important as the value of $k$ increases, and also if we build a k$^n$-tree for a high $n$, because each node of a k$^n$-tree will have $k^n$ children that must be represented if the region represented contains ones and zeros. Therefore, a k$^n$-tree may not be useful for unclustered data without any regularities. We will use the k$^n$-tree as an efficient method to represent multi-dimensional data in a way that ensures that the data is clustered across the different dimensions. Particularly, we will show its application to the representation of raster matrices.

## 5 Representation of Raster Data using k$^2$-tree Variants

In this section we design several structures for the representation of general raster data using the k$^2$-tree variants we have proposed in previous sections.

We assume in our proposals that the raster values have a "realistic" precision[6], so that the number of different values in the third dimension is not too

[6] When representing spatial attributes, in many cases the values stored can not be considered as an accurate estimation of actual values (e.g. a temperature measurement

high. We consider that the raster has $m$ different values and denote by $v_i$ the i-th different value of the raster in ascending order.

Our first representation of a raster dataset consists of a collection of k$^2$-trees (*k*$^2$*-base*), one for each different value of the raster (i.e., we build a k$^2$-tree $K_i$ that stores all the cells with value $v_i$). A variant with compression of ones will be used in order to exploit the regularities of spatial attributes. This representation can efficiently answer queries asking for cells with a given value, because these cells are indexed in a single k$^2$-tree. However, queries that ask for cells with values in a range $[v_\ell, v_r]$ require traversing many k$^2$-trees.

We also propose an alternative representation using "accumulated" k$^2$-trees (*k*$^2$*-acc*). In this representation, each k$^2$-tree $K_i$ will store not the cells whose value is $v_i$ but all the cells whose value is smaller or equal to $v_i$. While this increases the number of cells to represent in each k$^2$-tree, it will also increase the clustering of ones and therefore the ability of each k$^2$-tree to compress these regions. To ask for the value of a given cell in this variant, we can binary search the first k$^2$-tree that contains the desired cell of the raster. This variant also has an advantage when asking for cells with values within a given range, as we only need to query at most two k$^2$-trees: the cells with values in the range $[v_\ell, v_r]$ are the cells of $K_r$ that do not appear in $K_{\ell-1}$. As a counterpart, two k$^2$-trees must be queried instead of one in order to obtain the cells with a given value.

Finally, we propose a k$^3$-tree as a better representation of the raster data, as it is an indexed representation of the spatial coordinates and the values altogether. Our k$^3$-tree stores the tuples $\langle x, y, z \rangle$ such that the coordinate $(x, y)$ of the raster has value $z$. Notice that for each pair $(x, y)$ only one $z$ value will be set, so we will not find (cubic) regions of ones in this case. Because of this, our k$^3$-tree is based on the original k$^2$-tree codes, without compression of ones. To query for the value of a cell in the k$^3$-tree, we traverse all the paths found in the k$^3$-tree for fixed $x$ and $y$ values. To return all the cells with a given value, the $z$ coordinate is fixed and all the pairs $(x, y)$ of the corresponding k$^3$-tree slice are returned. To ask for cells with a range of values the query is similar, traversing the k$^3$-tree only within the bounds given by the interval $[z_\ell, z_r]$.

### 5.1 Experimental Framework

To test the efficiency of our proposals we use several real raster matrices obtained from the MDT05 collection. Table 3 gives details about the different fragments taken. The number of different values in each raster is also shown in the table, after rounding the elevation values to a precision of 1 meter.

To measure query results, we first determine a sufficiently large number of queries to obtain accurate results for each type of query. Then we build a different set of random queries of each type for each dataset. All the time results shown correspond to CPU time.

---

may be given in a precision of one thousandth of a degree, but this measurement may only be accurate in terms of a degree or a tenth of a degree).

| Dataset | Raster size | #values | Description |
|---|---|---|---|
| mdt-500 | 4001 × 5841 | 578 | Raster 500 |
| mdt-700 | 3841 × 5841 | 472 | Raster 700 |
| mdt-medium | 7721 × 11081 | 978 | Rasters 47,48, 72 and 73 combined |
| mdt-large | 48266 × 47050 | 2142 | Raster covering the region of Galicia |

**Table 3.** Raster datasets used.

### 5.2 Experimental Results

We show the space required to represent each dataset with all our proposals. To provide an element of comparison we convert the rasters to GeoTIFF[7] format using two different sets of options. The first, *tiff-plain*, is a plain representation without compression, that stores all the values in row order (we use 16-bit integers as the datatype for the representation). The second representation, *tiff-comp*, is optimized for space: the image is divided in tiles of size 256 × 256 and each tile is compressed using a linear predictor and LZW encoding.

Table 4 shows the space utilization of our proposals and the reference GeoTIFF images, in bits per cell of the raster. As an additional reference, columns 2 and 3 show the base-2 logarithm of the number of different values in each raster and the zero-order entropy $H_0$ of these values. These columns represent the minimum space that would be required by a representation of the raster as an uncompressed or entropy-compressed sequence, respectively. The k$^3$-tree clearly obtains the best space utilization amongst our approaches in all the datasets, being very close to the compressed GeoTIFF representation and using much less space than the zero-order entropy. Notice that the GeoTIFF format only offers, at best, random access to the raster data, whereas our proposals are indexed representations that efficiently solve various types of queries.

| Dataset | log(#values) | $H_0$(values) | k$^2$-base | k$^2$-acc | k$^3$-tree | tiff-plain | tiff-comp |
|---|---|---|---|---|---|---|---|
| mdt-500 | 9.17 | 5.43 | 2.83 | 2.30 | 1.83 | 16.01 | **1.52** |
| mdt-700 | 8.88 | 4.39 | 2.13 | 2.40 | 1.38 | 16.01 | **1.12** |
| mdt-medium | 9.93 | 5.86 | 3.06 | 2.72 | 1.77 | 16.01 | **1.52** |
| mdt-large | 11.06 | 5.32 | 3.16 | 4.37 | 1.62 | 16.00 | **1.35** |

**Table 4.** Space utilization of all approaches (in bits/cell).

Next we compare the query times of all our proposals for some queries of interest. We implement the same queries over the GeoTiff images used to compress the rasters. We build simple algorithms on top of the *libtiff* library[8] to retrieve fragments from the GeoTIFF images and run each query type. This comparison is given as a simple sanity check, since libtiff is not designed to process these queries[9]. We only aim to show that these queries are not easy to solve with the traditional formats.

[7] http://trac.osgeo.org/geotiff/

[8] http://www.libtiff.org

[9] The library *libtiff* reads the images from disk, but we are considering only CPU time.

First we measure the time required to retrieve the value of a single cell of the raster. This operation shows the ability of the representations to provide random access to the raster. Table 5 shows the results obtained. The $k^3$-tree representation obtains the best results among our proposals, showing the efficiency of the multi-dimensional index in this context. The approach based on independent $k^2$-trees, as expected, behaves much worse than our other proposals, since it has to scan the $k^2$-trees one by one. The accumulated $k^2$-tree approach obtains also good results because it can binary search the first $k^2$-tree that contains the cell. Note that the *tiff-plain* representation should obtain much faster times, but libtiff spends a lot of time copying chunks of the image that are useless for this query. On the other hand, the *tiff-comp* representation presents poor query times in comparison with the uncompressed version, as not only the appropriate tile of the image has to be recovered but also decompressed in order to recover a single cell.

| Dataset | $k^2$-base | $k^2$-acc | $k^3$-tree | tiff-plain | tiff-comp |
|---|---|---|---|---|---|
| mdt-500 | 66.7 | 4.6 | **2.2** | 2.6 | 491.7 |
| mdt-700 | 39.6 | 3.0 | **1.8** | 2.7 | 461.9 |
| mdt-medium | 76.4 | 5.3 | **2.6** | 5.2 | 499.0 |
| mdt-large | 415.3 | 11.1 | **2.8** | 87.9 | 494.8 |

**Table 5.** Retrieving the value of a single cell. Times in $\mu$s/query.

Next we show the efficiency of the representations to select the cells of the raster that contain a specific value. The results are shown in Table 6. Not surprisingly, our representations obtain much better results than the *tiff* representations, because the latter must always traverse the complete raster. In this case, the $k^2$-base obtains better results, as expected, because only one $k^2$-tree is accessed and the regions of the ones in the $k^2$-tree can be decoded efficiently. The $k^2$-acc has to access two $k^2$-trees, essentially doubling the query time of the independent $k^2$-trees. The $k^3$-tree only needs to traverse the appropriate slice, but it may need to explore many more nodes that correspond to regions with close values.

| Dataset | $k^2$-base | $k^2$-acc | $k^3$-tree | tiff-plain | tiff-comp |
|---|---|---|---|---|---|
| mdt-500 | **3.9** | 5.8 | 9.4 | 39.5 | 221.4 |
| mdt-700 | **3.0** | 6.0 | 7.3 | 37.5 | 199.5 |
| mdt-medium | **8.2** | 13.6 | 18.9 | 142.6 | 799.0 |
| mdt-large | **110.2** | 255.1 | 196.6 | 3,838.9 | 19,913.4 |

**Table 6.** Retrieving all the cells with a given value. Times in ms/query.

Finally, we measure the efficiency of window-range queries, that ask for cells of the raster within a spatial window and a range of values. These queries are widely used when processing raster data corresponding to spatial attributes (for instance, regions with risk of floods or snow alert may be computed from elevation rasters selecting the cells with values above a threshold or in a given

interval). In this case, the $k^3$-tree and the $k^2$-acc take advantage of their structure to obtain the best times. The $k^2$-acc only needs to perform a window query in two $k^2$-trees, and the $k^3$-tree can restrict the navigation of the tree in all the dimensions to the given bounds.

| Dataset | Window size | Range length | $k^2$-base | $k^2$-acc | $k^3$-tree | tiff-plain | tiff-comp |
|---|---|---|---|---|---|---|---|
| mdt-500 | 10 | 10 | 5.9 | **1.6** | 1.9 | 24.5 | 525.7 |
| | | 50 | 27.4 | **1.9** | 2.6 | 24.4 | 525.0 |
| | 50 | 10 | 10.3 | **3.6** | 5.1 | 124.0 | 697.1 |
| | | 50 | 51.0 | **5.4** | 16.2 | 124.0 | 699.5 |
| mdt-700 | 10 | 10 | 5.9 | **1.6** | **1.6** | 24.3 | 496.0 |
| | | 50 | 27.6 | **1.8** | 2.3 | 24.5 | 493.4 |
| | 50 | 10 | 10.1 | **3.6** | 4.5 | 123.7 | 653.2 |
| | | 50 | 49.2 | **5.0** | 13.6 | 123.8 | 649.9 |
| mdt-medium | 10 | 10 | 6.4 | 2.4 | **2.0** | 45.7 | 531.4 |
| | | 50 | 28.4 | **2.5** | **2.5** | 45.9 | 533.6 |
| | 50 | 10 | 9.9 | **3.7** | 4.2 | 229.2 | 705.4 |
| | | 50 | 46.5 | **4.7** | 10.9 | 228.5 | 705.4 |
| mdt-large | 10 | 10 | 10.5 | 3.9 | **2.2** | 285.5 | 519.1 |
| | | 50 | 44.2 | 3.9 | **2.5** | 287.4 | 545.8 |
| | 50 | 10 | 13.1 | 4.6 | **3.2** | 1,021.6 | 693.6 |
| | | 50 | 54.5 | **5.2** | 5.8 | 1,009.6 | 691.9 |

**Table 7.** Retrieving cells inside a window and within a range of values. Times in $\mu$s/query.

## 6 Conclusions

We have presented several compact data structures that can represent raster data in reduced space, supporting not only access to random areas in the raster but also advanced queries involving the values stored in the raster. We compare our representations, based on $k^2$-trees, with existing formats used to store and process raster data. Our experiments show that the $k^3$-tree can obtain very good space results, being close to the compressed GeoTIFF representation. The $k^3$-tree also shows competitive times in all the queries tested, being the fastest to retrieve the value of a cell and in some window queries. The variant with independent $k^2$-trees obtains the best time results to retrieve all the cells with a given value, but it is much slower in queries involving a range of values. The variant with accumulated $k^2$-trees obtains the best results in most of the queries involving ranges of values. In all the queries tested the results of our proposals are clearly better than the representations based on GeoTIFF images.

We believe that the proposed variants of the $k^2$-tree could be used in a wider range of application domains. We have shown, as a proof of concept, the applicability of the $k^2$-tree with compression of ones to the representation of binary images. Variants of $k^n$-tree could also be used to represent, for instance, spatio-temporal raster datasets and moving region databases. Spatio-temporal raster datasets can be seen as a collection of rasters stored for different time

instants, so we can consider the time as a fourth dimension in the matrix that represents the raster and use a k$^4$-tree to represent space, time and values stored. An example of spatio-temporal raster is a collection of temperature rasters in different days. In this example it is expected that the values stored for cells close in space or for the same cell along time are similar. Moving region databases represent regions of space that change with time. These regions can be encoded with a 3-dimensional matrix that stores spatial coordinates covered by a region along time, so that cells of the matrix determine if the region covered a given position at a given time. In many cases, these regions are continuous and change only slightly between time instants (e.g., the evolution of oil spills along time will yield a 3-dimensional matrix with large uniform regions that will change slowly with time). Therefore, a k$^3$-tree with compression of ones could exploit these regularities to obtain good compression results, providing also spatio-temporal query support.

## References


1. J. Barbay, T. Gagie, G. Navarro, and Y. Nekrich. Alphabet partitioning for compressed rank/select and applications. In *Proc. 21st ISAAC*, pages 315–326, 2010.
2. P. Boldi and S. Vigna. The Webgraph framework I: compression techniques. In *Proc. 13th WWW*, pages 595–602, 2004.
3. N. R. Brisaboa, G. de Bernardo, and G. Navarro. Compressed dynamic binary relations. In *Proc. 22nd DCC*, pages 52–61, 2012.
4. N. R. Brisaboa, S. Ladra, and G. Navarro. K$^2$-trees for compact Web graph representation. In *Proc. 16th SPIRE*, pages 18–30, 2009.
5. T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th SoCG*, pages 1–10, 2011.
6. H. K. Chang and J. W. Chang. Fixed binary linear quadtree coding scheme for spatial data. In *Proc. 9th VCIP*, volume 2308, pages 1214–1220, 1994.
7. R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.
8. I. Gargantini. An effective way to represent quadtrees. *Communications of the ACM*, 25(12):905–910, 1982.
9. A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proc. 17th SODA*, pages 368–373, 2006.
10. R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th SODA*, pages 841–850, 2003.
11. T. W. Lin. Set operations on constant bit-length linear quadtrees. *Pattern Recognition*, 30(7):1239–1249, 1997.
12. J. I. Munro. Tables. In *Proc. 16th FSTTCS*, pages 37–42, 1996.
13. G. Navarro. Wavelet trees for all. In *Proc. 23rd CPM*, pages 2–26, 2012.
14. P. Rigaux, M., and A. Voisard. *Spatial databases - with applications to GIS*. Elsevier, 2002.
15. T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.
16. M. Worboys and M. Duckham. *GIS: A Computing Perspective, 2nd Edition*. CRC Press, Inc., 2004.