Improved Queryable Representations of Rasters

Alejandro Pinto*, Diego Seco*, and Gilberto Gutiérrez†

*Universidad de Concepción
Concepción, Chile
{patriciopinto,dseco}@udec.cl

†Universidad del Bío-Bío
Chillán, Chile
ggutierr@ubiobio.cl

Abstract

We present two compact representations of rasters, which are used in GIS to represent temperatures, elevations, and other spatial attributes, that support queries on the positions and/or the values stored. These representations are based on space-filling curves and recent advances on compact data structures. They are practical, competitive with recent works on the problem, and present some improved characteristics, such as a nice generalization to time series of rasters, i.e. the storage of several rasters covering the same area at different times.

Introduction

The use of geographic data has become essential in decision making systems of many different domains, for example, in Smart Cities. Control surveillance, pollution control and weather forecasts are just some examples that require an efficient processing and management of geographic data. Geographic Information Systems (GIS) [1] have proven successful on providing effective solutions to problems involving geographic data. There exist two popular data models on GIS, namely vector and raster model, each of them tailored to different types of data. Whereas the vector model represents geographic objects such as roads, country borders, rivers, etc. using discrete primitives such as point, line and polygon; the raster model divides the space into a regular grid of cells with values, and it better represents elevation surfaces, pressures or temperatures, to name a few examples.

One of the main problems of the raster model is the space required to store the data. As the size of the cell defines the resolution of the raster, to store more accurate information more space is needed. Data compression has been used in the past to provide solutions to the storage and transmission of raster data [2, 3] and, GeoTIFF, which may be the gold standard to represent rasters, can be configured to use compression techniques such as Lempel-Ziv-Welch variants [4]. However, this only partially solves the problem as decision making systems usually need to query specific areas of these rasters. Again, GeoTIFF provides a partial solution to this problem as it decomposes the space into *tiles* that are compressed independently and thus, can be also retrieved independently. We named it a partial solution as some other types of queries such as retrieve the value of a specific cell, retrieve the values of any subregion (and not just tiles), or retrieve the values of a region that belong to a specific range of values are also useful and not efficiently supported by GeoTIFF.

This work was supported in part by MINECO (PGE & FEDER) TIN2013-46801-C4-3-R

In other words, space efficient representations of rasters with more advanced index capabilities are of interest. Specifically, we study compact representations of rasters for the three types of queries mentioned above which, in this paper, are named *access*, *windowQuery* and *rangeQuery*, respectively.

Many compact data structures [5] have been developed to provide query capabilities on compressed data without decompressing them, in different domains from bioinformatics to GIS. Recently, their use for rasters has been explored in [6, 7]. In [6] several data structures based on the k^2-tree [8] and its generalization to multidimensional data were proposed and compared with GeoTIFF, concluding that it is possible to support queries efficiently in space competitive with traditional compressed storage. Later, [7] showed that previous approaches do not scale well with the number of different values on the raster, and proposed a new method that is competitive when the raster contains few different values, but also scales much better both in space and query time. Their method, called k^2-raster, is also based on a k^2-tree, in which each node is augmented with additional data to speed-up queries on the values of the raster. These additional data are differentially encoded on the tree (similarly to [9]). Long before, Pajarola and Widmayer [10] had studied the idea of performing queries on compressed raster images. Their work is also previous to the era of the compact data structures as we know them today and it was developed for secondary memory. However they introduced some interesting ideas that we revisit with the new advances on the field. For example, the use of space-filling curves [11] to exploit spatial regularities based on Tobler's first law of geography *"everything is related to everything else, but near things are more related than distant things"*. However, unlike them, we use recent developments on compact data structures to provide efficient queriable representations of rasters on main memory.

Background

Space-filling curves are mathematical functions providing a mapping from a multidimensional space to one dimension [11], which have been heavily used on multidimensional indexes [12], specially on secondary memory, as they preserve locality on the original space, which helps in reducing the number of disk accesses on querying. Among all the existing curves, we use Morton (or Z-order) [13] as, besides preserving locality, it also has some nice properties for efficient computations. A simple bit interleaving of the binary codes of the indexes of a cell can be used to compute the Morton code in constant time (in theory and current hardware architectures).

Maximal Quadbox decomposition is the division of a general region into maximal boxes [14, 15] that would be obtained by representing the region using a Quadtree [12] or a contiguous range of the Z-order, giving the well-established relation between them. This decomposition can be obtained in linear time and it has been used to optimize queries in Quadtrees, ubiquitous on spatial databases.

Compact data structures [5] provide practical solutions to store and query large datasets, being a good complement to other approaches of the Big Data era such as distributed systems. As an example, a bitmap $B[1, n]$ can be stored in $n + o(n)$ bits supporting $rank_b(B, i)$ operations, i.e. count the number of occurrences of bit b in

$B[1, i]$, in constant time [16]. This solution is practical and it is also possible to reduce the space and still support some operations in constant time [17]. We also use succinct representations of trees [18] that require $2n + o(n)$ bits of space and provide constant time in many operations that are useful to traverse a tree. These and other examples are available in the SDSL library [19], which is used in our experiments.

k^2-trees [8] represent sparse binary matrices in compact space. The k^2-tree subdivides a matrix into k^2 submatrices, which are considered in left-to-right and top-to-bottom order, i.e. in Morton order, and each of them is represented with a 1-bit, if the submatrix is non-empty, or with a 0-bit, otherwise. Non-empty submatrices are recursively processed. This data structure can be devised as a space-efficient quadtree, hence the relationship with Morton order. The nodes of the k^2-tree can be augmented with additional data to support more advanced queries, such as aggregated queries [20]. To reduce the space overhead of the augmented data structure, these data can be differentially encoded using the same techniques proposed in [9]. A similar idea is described in [7] to provide query capabilities on space-efficient rasters.

Integer codes. There exist many techniques to encode sequences of integers in compressed space [21]. We use DACs [22], which are variable-length Codes with Direct Access. They can be described as a *Vbyte coding* in which the size of the *blocks* depends on the sequence and the blocks are reorganized in order to provide direct access to any position of the sequence, i.e. it is possible to retrieve the value at each position of the sequence in time that can be considered constant in practice.

Our solutions

In this section we describe two compact representations of rasters with support to the following operations: $access(x, y)$, which retrieves the value of the cell (x, y); $windowQuery(x_1, y_1, x_2, y_2)$, which retrieves all the values in the query window $[x_1, x_2] \times [y_1, y_2]$; and $rangeQuery(x_1, y_1, x_2, y_2, rMin, rMax)$, which is similar to the previous one, but it restricts the values to those contained in the range $[rMin, rMax]$.

First solution: 2D to 1D mapping

Our first proposal is a quite straightforward application of space-filling curves, revisited with recent developments of compact data structures. Space-filling curves have been used in combination with one dimensional indexes to support query capabilities, such as range queries, on geographic data. The idea is simple, a Morton curve defines an order in a set of geographic objects (vector objects or cells of a raster), which can be indexed by a 1D index such as a binary search tree (BST) or a B-tree. Then, a query on the original dataset is translated to a query on the 1D index. The details are explained below but, in a nutshell, we store the values obtained from a Morton order traversal of the raster in a Differentially Encoded Search Tree (DEST) [9].

Data structure. We read the values of the raster[1] on Morton order and generate a (virtual) sequence called *z-order-seq*, which is embedded in a BST on the positions

[1]For simplicity, we assume a square raster with side length a power of two. The general $n \times m$ case is extended to $2^{\log_2(max(n,m))}$ filling the new cells with zeros that are compressed later.

of the virtual sequence, and not on the values, using the algorithm described in [9]. As in [9], we distinguish the *tree representation*, i.e. the topology, from the *encoding* of the actual values. For the tree representation, we could rely on the binary heap embedding used by DEST as our mapping generates a complete binary tree and this embedding does not require any extra space. However, in order to achieve better compression some subtrees are pruned (thus creating a non-complete binary tree). The key observation is that a subtree represents a contiguous range of the Morton curve and, as this curve preserves locality, such range also represents close values on the original raster. Hence, by application of Tobler's first law of geography, those values are probably similar (for example, the temperatures of close cells –regions– are likely to be the same). When that is the case, we prune the tree and represent the whole subtree with just one node. Note that this is also the reason why the zeros introduced to generate a square grid do not cause a space overhead. This leads to a new problem as a binary heap embedding is not suitable to represent non-complete trees. A direct application of succinct trees [18] requires $2n + o(n)$ bits supporting a broad set of operations, but this space overhead may be significant in our application. Fortunately, our pruning generates trees in which each node has zero or two children, which can be represented in less space as stated in the following lemma.

Lemma 1 *A binary tree with n nodes, each of them having either zero or two children can be stored in $n + o(n)$ bits, supporting basic navigation operations in $O(1)$ time.*

A data structure for the above lemma may be constructed as follows. The tree is traversed in level order, writing down a 1-bit for each internal node and a 0-bit for each leaf. This generates a bitmap T_{bm} with n bits, which is augmented with support for rank operations (adding $o(n)$ bits of extra space). Each node is represented by its rank in level order, being the root at position 1. The left child of node i, if exists (i.e. if the bitmap contains a 1-bit at position i), is stored at position $2 \times rank(T_{bm}, i)$, whereas the right child is at position $2 \times rank(T_{bm}, i) + 1$. The parent of a node i is at position $\lfloor i/2 \rfloor$. All these operations can be computed in constant time. This data structure can be devised as a particular case of the LOUDS representation.

For the encoding of the values, as in [6, 7], we assume integer values in a range $[0, maxVal]$. There is also an important difference with the original DEST because the z-order-seq is not a monotone sequence and thus negative values may appear when applying the differential encoding (in [9], the value of a left child l of v is stored as $v - l$, and the value of a right child r is stored as $r - v$). To deal with this, we use the folklore zig-zag encoding, which represents a positive number i as $2i$ and a negative number $-i$ as $2i - 1$; for example, -3 is represented as $2 \times |-3| - 1 = 5$). This can be efficiently computed as $zz(i) = (i << 1) XOR (i >> w)$, where w is the word size. All these values are stored in a sequence T_{vals}, also in level order, which is represented with DACs [22], because it is expected to contain small values (due to the differential encoding) and the query algorithms require direct access to each position.

With T_{bm} and T_{vals}, the data structure supports queries on the positions of the raster, either individual cells or regions. In order to support queries on the values, each node can be augmented with a summary (min and max values) of the subregion

of the raster it represents. These values can be also differentially encoded, thus adding a small space overhead. In addition, it is possible to provide a space-time trade-off by augmenting just the nodes up to a certain depth of the tree (similarly to [20]). The details and evaluation of this extension will be in the full version of this article.

Query algorithms. The $access(x, y)$ operation is conceptually solved as a root to leaf traversal. First, the cell coordinates of the query are transformed to its corresponding Morton order. Each node v in the (conceptual) DEST represents a cell of the raster, i.e. its rank – in Morton order– and value. As this is a conceptual BST on the Morton codes, all the nodes in the left (alt. right) subtree of v represent cells with Morton order lower (alt. higher) than that of v. Hence, we can search this tree using the classical search algorithm of BSTs, i.e. traversing the left subtree if the queried position is lower than the position associated with the node and the right subtree, otherwise. This traversal can be implemented with the T_{bm} and T_{vals} components that actually form the data structure. The procedure starts at position 1 of T_{bm}, which represents the root of the tree. $T_{vals}[1]$ contains the value of its corresponding cell on the raster. Then, rank operations on T_{bm} are used to navigate the tree (as explained above). The corresponding positions on T_{vals} are differentially and zig-zag encoded with respect to their parents. Therefore, each time we move from a node to one of its children, the original value is decoded using the previous decoded value and the current value stored on T_{vals}. Recall that this sequence is stored using DACs, which provide direct access. The algorithm stops when it reaches a leaf, i.e. a 0-bit on T_{bm}, which represents a cell or a subregion of the raster filled with the value stored in such node. As all the operations can be supported in constant time, the complexity of this traversal is logarithmic in the size of the raster. For more details see [9][Algorithm 1], as this operation is similar to access on the original DEST.

The algorithm for the $windowQuery(x_1, y_1, x_2, y_2)$ operation is based on the properties of the Morton curve. Given the two query points defining the window, (x_1, y_1) and (x_2, y_2), all the cells of the original raster lying on such window are contained in the range of $z\text{-}order\text{-}seq[Z(x_1, y_1)..Z(x_2, y_2)]$, where $Z(x, y)$ represents the Morton code of cell (x, y). If the query window is a quadbox [14], such range contains exactly all the cells in the quadbox. However, for a general window, the range may contain some *false positives*, i.e. cells that are not inside the query window. We have studied two techniques to deal with general queries, which showed similar results on our preliminary evaluation. One is based on a quadbox decomposition [14, 15], in which case the problem reduces to support *windowQuery*s for quadboxes, which is explained below. The second one is based on the range search algorithm in [23], which detects when the Morton order traversal is outside the query window and computes the next element in such order lying inside the query window. In this paper, we focus on the first technique as it is simpler to explain and the performance is similar.

Hence, a simple algorithm to support $windowQuery(x_1, y_1, x_2, y_2)$ decomposes the general query into quadboxes (which takes linear time using [15]) and then processes each quadbox query, which can be efficiently solved using the following observation.

Observation 1 *A* $windowQuery(x_1, y_1, x_2, y_2)$ *where* $[x_1, x_2] \times [y_1, y_2]$ *defines a quad-*

box, can be solved via an in-order traversal of the tree from $Z(x_1, y_1)$ to $Z(x_2, y_2)$.

As the original DEST, this operation is supported in $O(\log n + l)$ time, where the $O(\log n)$ term comes from an $access(x_1, y_1)$ and l represents the size of the quadbox.

Second solution: 3D to 2D mapping

Our second solution is based on similar building blocks, but combined in a different way. Let us consider 3D tuples $< x, y, z >$, with (x, y) representing a cell in the raster and z the value of such cell (again an integer value in $[0, maxVal]$). These tuples are mapped into a 2D binary grid, BG, in which the X-axis represents positions in Morton order and the Y-axis represents values. Then, for each tuple $< x, y, z >$, there is a 1-bit at position $BG[Z(x, y)][z]$ and zeros in rest of the matrix. For example, given the tuple $< 1, 1, 4 >$, which indicates that the value of the cell $(1, 1)$ in the raster is 4, then $BG[Z(1, 1)][4] = BG[3][4] = 1$.

Summarizing the properties of BG, given a raster of size $n \times m$ with values in $[0, maxVal]$, i) BG is a binary matrix with $n \times m$ columns and $maxVal + 1$ rows, ii) there is just one cell at each column containing a 1-bit, iii) as Morton order preserves spatial locality in the original space, contiguous columns on BG are expected to contain 1-bits on near rows (in other words, the 1-bits are expected to be clustered), iv) the three types of queries studied in this article can be solved by range queries on BG as we explain below.

Data structure and query algorithms. Given the properties described above, any compact data structure for binary grids supporting range queries may suit our purposes. Specifically, we propose the use of a k^2-tree. (We also evaluated the use of a compressed Wavelet tree, which is faster for some queries, but uses more space).

To explain the query algorithms, we will use the same names as in the description of the original k^2-tree [8], although all of them are variants or special cases of range queries. An $access(x, y)$ operation is then solved using $Predecessors(Z(x, y))$ (see [8][Algorithm 4]), which retrieves all the ones in a column (in our case, the only 1-bit in the column). The queried column is previously obtained by computing the Morton code of the original coordinates x, y (i.e. $Z(x, y)$).

The other two queries require a quadbox decomposition as in our first data structure. Once the query window has been decomposed into quadboxes, each quadbox query can be solved as follows. A $windowQuery(x_1, y_1, x_2, y_2)$ is solved using $Range(Z(x_1, y_1), Z(x_2, y_2), 0, maxVal)$. This retrieves all the ones in a contiguous range of columns. On the other hand, $rangeQuery(x_1, y_1, x_2, y_2, rMin, rMax)$ is solved as $Range(Z(x_1, y_1), Z(x_2, y_2), rMin, rMax)$, which is similar to the previous one, but restricts the ones to a specific range of rows. For details see [8][Algorithm 5].

Bonus: Extension to raster time series. In many domains, a set of rasters covering the same region at different times needs to be stored and queried. This is the case, for example, in weather forecast systems and in data mining on *Satellite Image Time Series* [24]. A naive approach is to represent and query each raster

separately using any of the techniques evaluated in this article. However, the 3D2D-map provides a nice generalization to this problem with additional advantages.

The idea is to consider the set of rasters in time order. Then, apply the 3D to 2D transformation to each raster, which generates a 2D binary grid. Concatenate all these grids into a 3D binary grid and represent it with a k^3-tree (a 3D version of the k^2-tree). By doing this, we can exploit not only the spacial locality, but also the temporal locality. In simple words, the 1-bits will not only be clustered at each slice (i.e. each 2D grid), but also through slices. This is similar to the representation of moving regions, i.e. a generalization of moving objects, in which the objects have a certain area. We conjecture that this representation requires less space than the individual compression of each raster independently. In addition, it would also support interesting queries for data mining as it represents all the rasters in the same data structure. This extension will be explored in the full version of this article.

Experiments

As a proof of concept, we have implemented the solutions described above and compared them experimentally with the compact representations described in [6, 7], which source code was kindly provided by the authors. A comparison with more traditional techniques, such as GeoTIFF, was already presented in [6], hence we omit such comparison here for briefness. All the experiments presented here were performed in an Intel Core i7-3820@3.60GHz, 32GB RAM, running Ubuntu server (kernel 3.13.0-35). We compiled with gnu/g++ version 4.6.3 using -O3 directive as all the data structures, including the baselines, are in C++. For the baselines, we use the configuration parameters recommended by the authors on their original papers.

Regarding the datasets, as in [6, 7], we use several Digital Terrain Models (DTMs) from the Spanish Geographic Institute (`http://www.ign.es`) with a resolution of 5 meters. Due to space constraints, we present the results with just two of such datasets, MDT-500 and MDT-700 (the latter just in tables, but not in figures).

Table 1 shows a comparison of the space used by the different proposals, where 2D1D-map and 3D2D-map are our two proposals, the k^3-tree and k^2-tree Acum are described in [6], and the k^2-raster is described in [7].

Dataset	2D1D-map	3D2D-map	k^3-tree	k^2-tree Acum	k^2-raster$_H$
MDT-500	2.75	2.61	1.83	2.30	2.82
MDT-700	1.89	2.15	1.38	2.40	1.87

Table 1: Space usage in bits per cell.

The space usage of all the variants is similar, being the k^3-tree the most space-efficient. Note, however, that within this space, the 2D1D-map just supports queries on the positions (*access* and *windowQuery*) and it would need some extra space to support *rangeQuery*. Based on [20], this space could range from an extra 1% to 20%. Also, as claimed in [7], the performance of the k^3-tree (both in space and query time) does not scale well with the number of different values in the raster. To show this, we use six datasets derived from the MDT05-0533-H30. Each of them corresponds with

a shifting of x bits (with $x = 1, 3, 5, 7, 9$) in the values of the raster. This generates rasters of the same size but with different number of different values (see Table 2).

Dataset	#diff values	2D1D-map	3D2D-map	k^3-tree	k^2-raster$_H$
MDT05-0533-H30$_{>>9}$	227	1.26	1.73	1.01	1.21
MDT05-0533-H30$_{>>7}$	903	2.94	2.94	2.91	3.07
MDT05-0533-H30$_{>>5}$	3,606	5.24	6.71	7.33	6.01
MDT05-0533-H30$_{>>3}$	14,415	7.35	10.83	10.25	9.16
MDT05-0533-H30$_{>>1}$	57,586	9.36	13.80	-	12.39
original	114,966	10.36	16.07	-	13.76

Table 2: Space usage in bits per cell.

As in [7], we were not able to run the experiment with the k^2-tree Acum. For the k^3-tree, we obtained results up to 14 thousand different values in the raster. In conclusion, our two proposals and the k^2-raster scale much better in the number of different values (this is for space, but the same trend can be observed in query time).

Regarding query time, Table 3 shows a comparison in terms of query time to solve an *access* operation. For each dataset, a query-set with $1,000$ random queries was generated. We run the experiment 20 times and show the average time per query. A similar methodology is used in following experiments.

Dataset	2D1D-map	3D2D-map	k^3-tree	k^2-raster$_H$
MDT-500	4.60	1.40	2.20	0.80
MDT-700	3.40	1.20	1.60	0.60

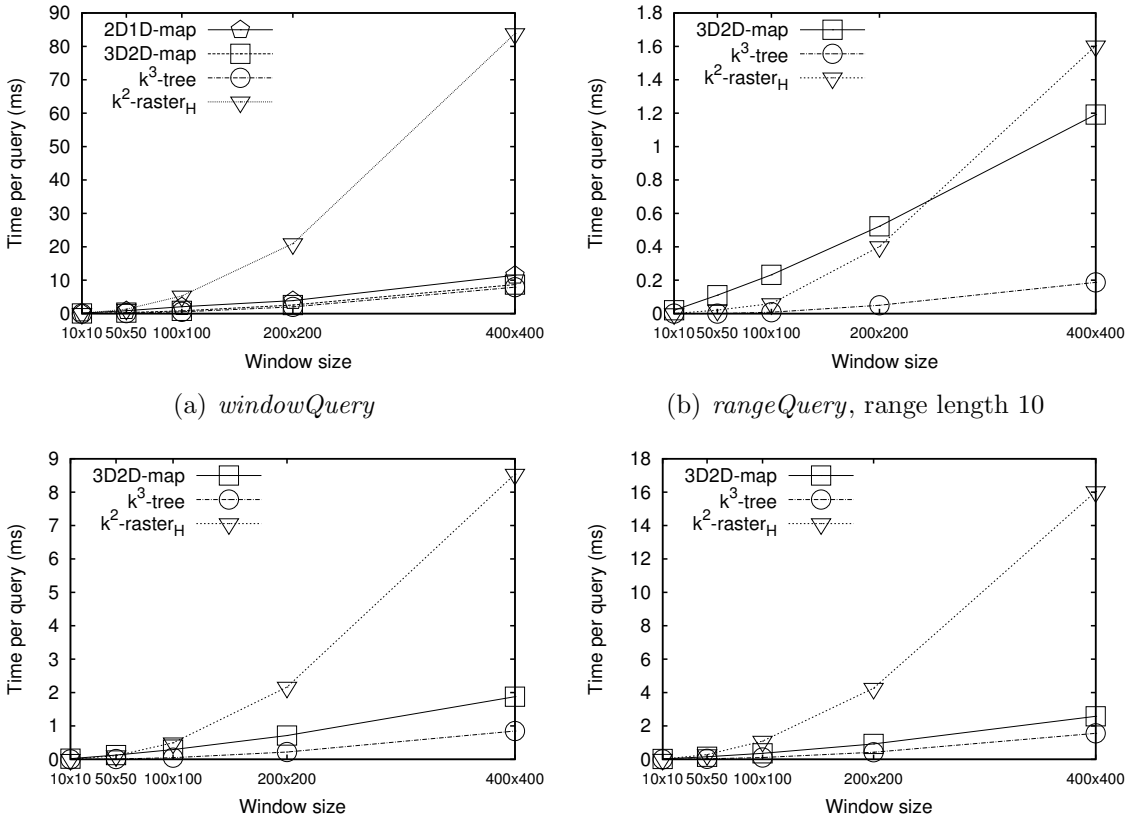
Table 3: Query time for *access* in microseconds (μs).

The performance is similar and just the 2D1D-map is significantly slower than the others but still comparable. Hereinafter, the k^2-tree Acum is not included as we were not able to run the code. However, from the evaluations in [6, 7], it is always outperformed by either the k^3-tree or the k^2-raster.

For *windowQuery*, Figure 1(a) shows a comparison for different window sizes (X-axis). These results show that both the $2D1D$-map and $3D2D$-map are competitive with the k^3-tree and much better than the k^2-raster for large queries. However, we must clarify that the source code provided by the authors computes the positions of the raster satisfying the query and not their actual value. We adapted their code by running an *access* for each returned position, which may be inefficient.

Similar results are shown in Figures 1(b),1(c),1(d) for *rangeQuery*. The larger the window query (represented in X-axis) or the range of values (in each graph), the better the performance of the $3D2D$-map. Recall that we did not implemented this operation on the $2D1D$-map. For larger ranges, the performance becomes similar to *windowQuery* (Figure 1(a)), which is a special case retrieving all the different values.

Finally, a particular case of range queries is when the query window is a quadbox. This may be of interest when the queries are defined by the system and not by the user, for example, *tiles* in GIS. In this case, the $3D2D$-map does not have to perform the quadbox decomposition (and perform a range query for each quadbox) and it outperforms all the other data structures for all query size.



(a) *windowQuery*

(b) *rangeQuery*, range length 10

(c) *rangeQuery*, range length 50

(d) *rangeQuery*, range length 100

Figure 1: Query time performance of *windowQuery* and *rangeQuery*.

Conclusions

We presented two data structures to store rasters in compact space supporting queries. These representations are based on space-filling curves and compact data structures. Preliminary results show that they are practical and competitive both in space and time, although not markedly better, than the state of the art. The $2D1D$-map is composed of two structures, one for the positions and the other for the values, which is not evaluated in this paper. Hence, it has the advantage of using less space when queries on the values are not required. The $3D2D$-map performs the best when the queries are large (either on the size of the query window or the range of values). In addition, when the query windows are quadboxes, our two proposals outperform the state of the art. A complete evaluation will be presented in the full version of this article. We will also explore the use of more sophisticated compressed Wavelet trees [25] on the $3D2D$-map, and its generalization to domains in which the same region has to be represented at different timestamps, which we called raster time series. This is promising as it may improve the compression ratio with respect to the individual compression of each raster and has applications in data mining.

References

[1] M. Worboys and M. Duckham, *GIS: A Computing Perspective*. CRC Press, Inc., 2004.

[2] W. Kou, *Digital Image Compression: Algorithms and Standards*. Kluwer Pub., 1995.

[3] G. K. Wallace, "The jpeg still picture compression standard," *Commun. ACM*, vol. 34, no. 4, pp. 30–44, 1991.

[4] D. Salomon, *Data Compression: The Complete Reference*. Springer, 2006.

[5] G. Navarro, "Compact data structures: A practical approach," 2016.

[6] G. de Bernardo, S. lvarez Garca, N. R. Brisaboa, G. Navarro, and O. Pedreira, "Compact querieable representations of raster data," in *Proc. 20th SPIRE*, 2013, pp. 96–108.

[7] S. Ladra, J. R. Paramá, and F. Silva Coira, "Compact and queryable representation of raster datasets," in *Proc. 28th SSDBM*, 2016.

[8] N. R. Brisaboa, S. Ladra, and G. Navarro, "Compact representation of web graphs with extended functionality," *Information Systems*, pp. 152–174, 2014.

[9] F. Claude, P. K. Nicholson, and D. Seco, "On the compression of search trees," *Information Processing and Management*, pp. 272–283, 2014.

[10] R. Pajarola and P. Widmayer, "An image compression method for spatial search," *Trans. Img. Proc.*, vol. 9, no. 3, pp. 357–365, Mar. 2000.

[11] H. Sagan, *Space-Filling Curves*. Springer, 1994.

[12] H. Samet, *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[13] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," Tech. Rep., 1966.

[14] G. Proietti, "An optimal algorithm for decomposing a window into maximal quadtree blocks," *Acta Informatica*, vol. 36, no. 4, pp. 257–266, 1999.

[15] Y.-H. Tsai, K.-L. Chung, and W.-Y. Chen, "A strip-splitting-based optimal algorithm for decomposing a query window into maximal quadtree blocks," *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, no. 4, pp. 519–523, Apr. 2004.

[16] G. Jacobson, "Space-efficient static trees and graphs," in *Proc. SFCS*, 1989, pp. 549–554.

[17] R. Raman, V. Raman, and S. Rao, "Succinct indexable dictionaries with applications to encoding k-ary trees and multisets," in *Proc. SODA*, 2002, pp. 233–242.

[18] G. Navarro and K. Sadakane, *Compressed Tree Representations*, 2nd ed. Springer, 2016, pp. 397–401.

[19] S. Gog, T. Beller, A. Moffat, and M. Petri, "From theory to practice: Plug and play with succinct data structures," in *Proc. 13th SEA*, 2014, pp. 326–337.

[20] N. R. Brisaboa, G. De Bernardo, R. Konow, G. Navarro, and D. Seco, "Aggregated 2d range queries on clustered points," *Information Systems*, vol. 60, pp. 34–49, 2016.

[21] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*. San Francisco, CA, USA: Morgan Kaufmann, 1999.

[22] N. R. Brisaboa, S. Ladra, and G. Navarro, "Dacs: Bringing direct access to variable-length codes," *Information Processing and Management*, pp. 392–404, 2013.

[23] H. Tropf and H. Herzog, "Multimensional range search in dynamically balanced trees," *Angewandte Informatik*, vol. 23, no. 2, pp. 71–77, 1981.

[24] F. Petitjean, P. Gançarski, F. Masseglia, and G. Forestier, *Analysing Satellite Image Time Series by Means of Pattern Mining*. Springer Berlin Heidelberg, 2010, pp. 45–52.

[25] G. Navarro, S. J. Puglisi, and D. Valenzuela, "General document retrieval in compact space," *ACM Journal of Experimental Algorithmics*, vol. 19, no. 2, 2014.