Towards a compact representation of temporal rasters [⋆]

Ana Cerdeira-Pena[1], Guillermo de Bernardo[1,2], Antonio Fariña[1], José R. Paramá[1], and Fernando Silva-Coira[1]

[1] Universidade da Coruña, Fac. Informática, CITIC, Spain
[2] Enxenio S.L.

Abstract. Big research efforts have been devoted to efficiently manage spatio-temporal data. However, most works focused on vectorial data, and much less, on raster data. This work presents a new representation for raster data that evolve along time named `Temporal k`^{2}`raster`. It faces the two main issues that arise when dealing with spatio-temporal data: the space consumption and the query response times. It extends a compact data structure for raster data in order to manage time and thus, it is possible to query it directly in compressed form, instead of the classical approach that requires a complete decompression before any manipulation. In addition, in the same compressed space, the new data structure includes two indexes: a spatial index and an index on the values of the cells, thus becoming a self-index for raster data.

1 Introduction

Spatial data can be represented using either a raster or a vector data model [6]. Basically, vector models represent the space using points and lines connecting those points. They are used mainly to represent man-made features. Raster models represent the space as a tessellation of disjoint fixed size tiles (usually squares), each one storing a value. They are traditionally used in engineering, modeling, and representations of real-word elements that were not made by men, such as pollution levels, atmospheric and vapor pressure, temperature, precipitations, wind speed, land elevation, satellite imagery, etc.

Temporal evolution of vectorial data has been extensively studied, with a large number of data structures to index and/or store spatio-temporal data. Examples are the 3DR-tree [14], HR-tree [10], the MVR-tree [13], or PIST [3].

In [9] the classical Map Algebra of Tomlin for managing raster data is extended to manage raster data with a temporal evolution. The conceptual solution

⋆ Funded in part by European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 690941 (project BIRDS); by Xunta de Galicia/FEDER-UE [CSI: ED431G/01 and GRC: ED431C 2017/58]; by MINECO-AEI/FEDER-UE [Datos 4.0: TIN2016-78011-C4-1-R; Velocity: TIN2016-77158-C4-3-R; and ETOME-RDFD3: TIN2015-69951-R]; and by MINECO-CDTI/FEDER-UE [INNTERCONECTA: uForest ITC-20161074].

is simple, instead of considering a matrix, it considers a cube, where each slice of the temporal dimension is the raster corresponding to one time instant.

Most real systems capable of managing raster data, like Rasdaman, Grass, or even R are also capable of managing time-series of raster data. These systems, as well as raster representation formats such as NetCDF (standard format of the OGC[3]) and GeoTiff, rely on classic compression methods such as run length encoding, LZW, or Deflate to reduce the size of the data. The use of these compression methods poses an important drawback to access a given datum or portion of the data, since the dataset must be decompressed from the beginning.

Compact data structures [7, 11] are capable of storing data in compressed form and enable us to access a given datum without the need for decompressing from the beginning. In most cases, compact data structures are equipped with an index that provides fast access to data. There are several compact data structures designed to store raster data [2, 8]. In this work, we extend one of those compact data structures, the $\mathsf{k^2raster}$ [8], to support representing time-series of rasters.

2 Related work

In this section, we first revise the $\mathsf{k^2tree}$, a compact data structure that can be used to represent binary matrices. Then, we also present several compact data structures for representing raster data containing integers in the cells. We pay special attention to discuss one of them, the $\mathsf{k^2raster}$, which is the base of our proposal `Temporal` $\mathsf{k^2raster}$ ($\mathsf{T-k^2raster}$).

$\mathsf{k^2tree}$: The $\mathsf{k^2tree}$ [5] was initially designed to represent web graphs, but it also allows to represent binary matrices, that is, rasters where the cells contain only a bit value. It is conceptually a non-balanced k^2-ary tree built from the binary matrix by recursively dividing it into k^2 submatrices of the same size. First, the original matrix is divided into k^2 submatrices of size n^2/k^2, being $n \times n$ the size of the matrix. Each submatrix generates a child of the root whose value is 1 if it contains at least one 1, and 0 otherwise. The subdivision continues recursively for each node representing a submatrix that has at least one 1, until the submatrix is full of 0s, or until the process reaches the cells of the original matrix (i.e., submatrices of size 1×1).

The $\mathsf{k^2tree}$ is compactly stored using just two bitmaps T and L. T stores all the bits of the conceptual $\mathsf{k^2tree}$, except the last level, following a level-wise traversal: first the bit values of the children of the root, then those in the second level, and so on. L stores the last level of the tree.

It is possible to obtain any cell, row, column, or window of the matrix very efficiently, by running *rank* and *select* operations [7] over the bitmaps T and L.

$\mathsf{k^3tree}$: The $\mathsf{k^3tree}$ [2] is obtained by simply adding a third dimension to the $\mathsf{k^2tree}$, and thus, it conceptually represents a binary cube. This can be trivially done by using the same space partitioning and representation techniques from the $\mathsf{k^2tree}$, yet applied to cubes rather than to matrices.

[3] http://www.opengeospatial.org/standards/netcdf

Thus, each 1 in the binary cube represents a tuple $\langle x, y, z \rangle$, where (x, y) are the coordinates of the cell of the raster and z is the value stored in that cell.

k^{2}acc: The k^{2}acc [2] representation for a raster dataset is composed by as many k^{2}trees as different values can be found in the raster. Given t different values in the raster: $v_1 < v_2 < \ldots < v_t$, k^{2}acc contains $K_1, K_2, \ldots, K_t$ k^{2}trees, where each K_i has a value 1 in those cells whose value is $v \leq v_i$.

2D-1D mapping: In [12], it is presented a method that uses a space-filling curve to reduce the raster matrix to an array, and the use of one dimensional index (for example a B-tree) over that array to access the data.

k^{2}raster: k^{2}raster has proven to be superior in both space and query time [12, 8] to all the other compact data structures for storing rasters. In [8], it was also compared with NetCDF. It drew slightly worse space needs than the compressed version (that uses Deflate) of NetCDF, but queries performed noticeably faster.

k^{2}raster is based in the same partitioning method of the k^{2}tree, that is, it recursively subdivides the matrix into k^2 submatrices and builds a conceptual tree representing these subdivisions. Now, in each node, instead of having a single bit, it contains the minimum and maximum values of the corresponding submatrix. The subdivision stops when the minimum and maximum values of the submatrix are equal, or when the process reaches submatrices of size $1{\times}1$. Again the conceptual tree is compactly represented using, in addition to binary bitmaps, efficient encoding schemes for integer sequences.

More in detail, let $n{\times}n$ be the size of the input matrix. The process begins by obtaining the minimum and maximum values of the matrix. If these values are different, they are stored in the root of the tree, and the matrix is divided into k^2 submatrices of size n^2/k^2. Each submatrix produces a child node of the root storing its minimum and maximum values. If these values are the same, that node becomes a leaf, and the corresponding submatrix is not subdivided anymore. Otherwise, this procedure continues recursively until the maximum and minimum values are the same, or the process reaches a $1{\times}1$ submatrix.

Figure 1 shows an example of the recursive subdivision (top) and how the conceptual tree is built (centre-top), where the minimum and maximum values of each submatrix are stored at each node. The root node corresponds to the original raster matrix, nodes at level 1 correspond to submatrices of size $4{\times}4$, and so on. The last level of the tree corresponds to cells of the original matrix. Note, for instance, that all the values of the bottom-right $4{\times}4$ submatrix are equal; thus, its minimum and maximum values are equal, and it is not further subdivided. This is the reason why the last child of the root node has no children.

The compact representation includes two main parts. The first one represents the topology of the tree (T) and the second one stores the maximum/minimum values at the nodes ($Lmin/Lmax$). The topology is represented as in the k^{2}tree, except that the last level (L) is not needed. The maximum/minimum values are differentially encoded with respect to the values stored at the parent node. Again, these values are stored as arrays following the same method of the k^{2}tree, that is, following the same level-wise order of the conceptual tree. By using differential

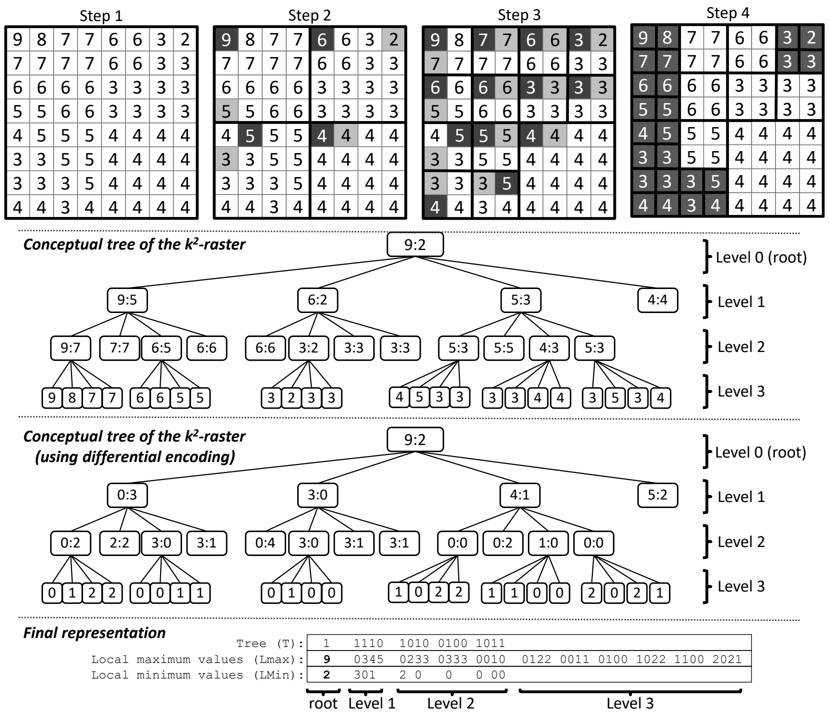

Step 1
Step 2
Step 3
Step 4
Conceptual tree of the k²-raster
Level 0 (root)
Level 1
Level 2
Level 3
Conceptual tree of the k²-raster (using differential encoding)
Final representation
Tree (T): 1 1110 1010 0100 1011
Local maximum values (Lmax): 9 0345 0233 0333 0010 0122 0011 0100 1022 1100 2021
Local minimum values (LMin): 2 301 2 0 0 0 00
root
Level 1
Level 2
Level 3

Fig. 1. Example (using $k = 2$) of integer raster matrix (top), conceptual tree of the k^{2}raster, conceptual tree with differential encoding, and final representation of the raster matrix. *Lmax* and *Lmin* contain the maximum and minimum values of each node following a level-wise order and using differential encoding.

encoding, the numbers become smaller. *Directly Addressable Codes* (DACs) [4] take advantage of this, and at the same time, provide direct access. The last two steps to create the final representation of the example matrix are also illustrated in Figure 1. In the center-bottom and bottom parts, we respectively show the tree with the differences for both the maximum and minimum values, and the data structures that compose the final representation of the k^{2}raster. Therefore, the original raster matrix is compactly stored using just a bitmap T, which represents the tree topology, and a pair of integer arrays (*Lmax* and *Lmin*), which contain the minimum and maximum values stored at the tree. Note that when the raster matrix contains uniform areas, with large areas of equal or similar values, this information can be stored very compactly using differential encoding and DACs.

The maximum/minimum values provide indexation of the stored values, this technique is usually known as lightweight indexation. It is possible to query the structure only decompressing the affected areas. Queries can be efficiently computed navigating the conceptual tree by running *rank* and *select* operations on T and, in parallel, accessing the arrays *Lmax* and *Lmin*.

3 T−k²raster: A temporal representation for raster data

Let $\mathcal{M}$ be a raster matrix of size $n \times n$ that evolves along time with a timeline of size τ time instants. We can define $\mathcal{M} = \langle M_1, M_2, \ldots, M_\tau \rangle$ as the sequence of raster matrices M_i of size $n \times n$ for each time instant $i \in [1, \tau]$.

A rather straightforward baseline representation for the temporal raster matrix $\mathcal{M}$ can be obtained by simply representing each raster matrix M_i in a compact way with a k²raster. In this section we use a different approach. The idea is to use sampling at regular intervals of size t_δ. That is, we represent with a k²raster all the raster matrices $M_s, s = 1 + i\ t_\delta,\ i \in [0, (\tau - 1)/t_\delta]$. We will refer to those M_i rasters as *snapshots of $\mathcal{M}$ at time i*. The $t_\delta - 1$ raster matrices $M_t, t \in [s+1, s+t_\delta - 1]$ that follow a snapshot M_s are encoded using M_s as a reference. The idea is to create a modified k²raster′ to represent M_t where, at each step of the construction process, the values in the submatrices are encoded as differences with respect to the corresponding submatrices in M_s rather than as differences with respect to the parent node as usual in a regular k²raster.

With this modification, we still expect to encode small gaps for the maximum and minimum values in each node of the conceptual tree of M_t. Yet, in addition, when a submatrix in M_t is identical to the same submatrix in M_s, or when all the values in both submatrices differ only in a unique gap value α, we can stop the recursive splitting process and simply have to keep a reference to the corresponding submatrix of M_s and the gap α (when they are identical, we simply set $\alpha = 0$). In practice, keeping that reference is rather cheap as we only have to mark, in the conceptual tree of M_t, that the subtree rooted at a given node p has the same structure of the one from the conceptual tree of M_s. For such purpose, in the final representation of k²raster′, we include a new bitmap eqB, aligned to the zeroes in T. That is, if we have $T[i] = 0$ (node with no children), we set $eqB[rank_0(T,i)] \leftarrow 1$,[4] and set $Lmax[i] \leftarrow \alpha$. Also, if we have $T[i] = 0$, we also can set $eqB[rank_0(T,i)] \leftarrow 0$ and $Lmax[i] \leftarrow \beta$ (where β is the gap between the maximum values of both submatrices) to handle the case in which the maximum and minimum values in the corresponding submatrix are identical (as in a regular k²raster).

The overall construction process of the k²raster′ for the matrix M_t related to the snapshot M_s can be summarized as follows. At each step of the recursive process, we consider a submatrix of M_t and the related submatrix in M_s. Let the corresponding maximum and minimum values of the submatrix of M_t be max_t and min_t, and those of M_s be max_s and min_s respectively. Therefore:

- If max_t and min_t are identical (or if we reach a 1×1 submatrix), the recursive process stops. Being z_t the position in the final bitmap T, we set $T[z_t] \leftarrow 0$, $eqB[rank_0[T, z_t]] \leftarrow 0$, and $Lmax[z_t] \leftarrow (max_t - max_s)$.[5]

[4] From now on, asume $rank_b(B,i)$ returns the number of bits set to b in $B[0, i-1]$, and $rank_b(B,0) = 0$. Note that the first index of T, eqB, $Lmax$, and $Lmin$ is 0.

[5] Since in k²raster′ we have to deal both with positive and negative values, we actually apply a *zig-zag* encoding for the gaps $(max_t - max_s)$.

– If all the values in M_s and M_t differ only in a unique value α (or if they are identical, hence $\alpha = 0$), we set $T[z_t] \leftarrow 0$, $eqB[rank_0[T, z_t]] \leftarrow 1$, and $Lmax[z_t] \leftarrow (max_t - max_s)$.
– Otherwise, we split the submatrix M_t into k^2 parts and continue recursively. We set $T[z_t] \leftarrow 1$, and, as in the regular k^{2}raster, $Lmax[z_t] \leftarrow (max_t - max_s)$, and $Lmin[rank_1(z_t)] \leftarrow (min_t - min_s)$.

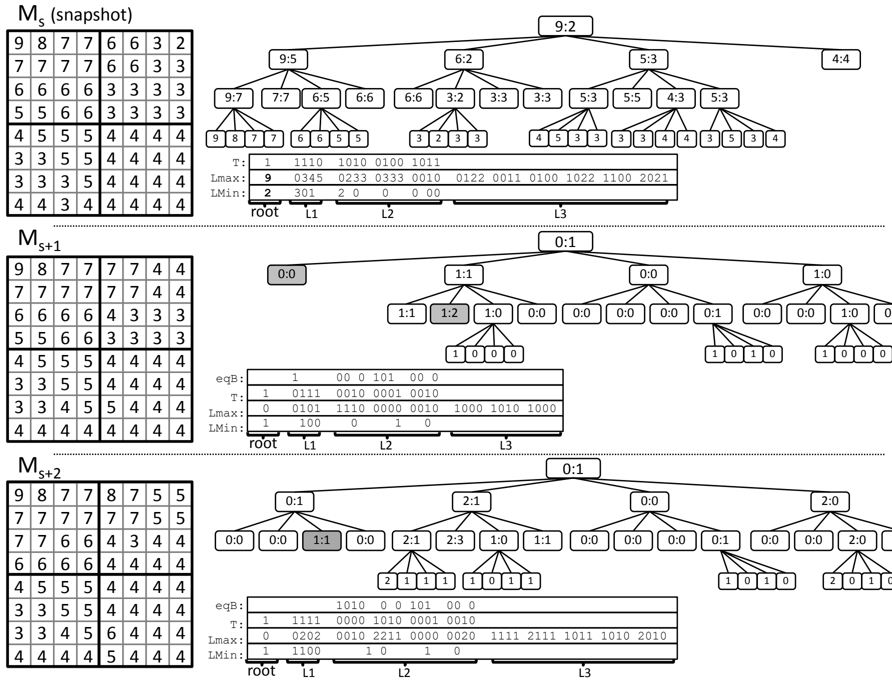


Fig. 2. Structures involved in the creation of a T$-$k^{2}raster considering $\tau = 3$.

Figure 2 includes an example of the structures involved in the construction of a T$-$k^{2}raster over a temporal raster of size 8×8, with $\tau = 3$. The raster matrix corresponding to the first time instant becomes a *snapshot* that is represented exactly as the k^{2}raster in Figure 1. The remaining raster matrices M_{s+1} and M_{s+2} are represented with two k^{2}raster$'$ that are built taking M_s as a reference. We have highlighted some particular nodes in the differential conceptual trees corresponding to M_{s+1} and M_{s+2}. *(i)* the shaded node labeled $\langle 0{:}\,0 \rangle$ in M_{s+1} indicates that the first 4×4 submatrix of both M_s and M_{s+1} are identical.

eqB está alineado con 0s en T
eqB = 0 si min y máx iguales en el nodo
eqB =1, mismos valores que snapshot

Therefore, node $\langle 0{:}\,0 \rangle$ has no children, and we set: $T[2] \leftarrow 0$, $eqB[1] \leftarrow 1$, and $Lmax[2] \leftarrow 0$. *(ii)* the shaded node labeled $\langle 1{:}\,1 \rangle$ in M_{s+2} illustrates the case in which all the values of a given submatrix are increased by $\alpha \leftarrow 1$. In this case values $\langle 6, 6, 5, 5 \rangle$ in M_s become $\langle 7, 7, 6, 6 \rangle$ in M_{s+2}. Again, the recursive traversal stops at that node, and we set: $T[8] \leftarrow 0$, $eqB[3] \leftarrow 1$, and $Lmax[8] \leftarrow 1$ (values are increased by 1). *(iii)* the shaded node labeled $\langle 1{:}\,2 \rangle$ in M_{s+1} corresponds to the node labeled $\langle 3{:}\,2\rangle$ in M_s. In this case, when we sum the maximum and minimum values of both nodes we obtain that that node in M_{s+1} has the same maximum and minimum values (set to 4). Consequently the recursive process stops again. In this case, we set $T[7] \leftarrow 0$, $eqB[3] \leftarrow 0$, and $Lmax[7] \leftarrow 1$.

4 Querying temporal raster data

In this section, we show two basic queries over $\mathsf{T{-}k^2raster}$.

Obtaining a cell value in a time instant: This query retrieves the value of a cell (r, c) of the raster at time instant t: $v \leftarrow getCellValue(r, c, t)$. For solving this query, there are two cases: if t is represented by a snapshot, then the algorithm to obtain a cell in the regular $\mathsf{k^2raster}$ is used, otherwise, a synchronized top-down traversal of the trees representing that time instant (M_t) and the closest previous snapshot (M_s) is required.

Focusing on the second case, the synchronized traversal inspects the two nodes at each level corresponding to the submatrix that contains the queried cell. The problem is that due to parts of M_t or M_s having the same value, the shape of the trees representing them can be different. Therefore, it is possible that one of the two traversals reaches a leaf, whereas the other does not. In such a case, the traversal that did not reach a leaf, continues, but the process must remember the value in the reached leaf, since that is the value that will be added or subtracted to the value found when the continued traversal reaches a leaf.

Indeed, we have three cases: (a) the processed submatrix of M_t is uniform, (b) the original submatrix of M_s is uniform and, (c) the processed submatrix after applying the differences with the snapshot has the same value in all cells.

Algorithm 1 shows the pseudocode of this case. To obtain the value stored at cell (r, c) of the raster matrix M_t, it is invoked as **getCell**$(n, r, c, 1, 1, Lmax_s[0], Lmax_t[0])$, assuming that the cell at position (0,0) of the raster is that in the upper-left corner.

z_s is used to store the current position in the bitmap T of M_s (T_s) during the downward traversal at any given step of the algorithm, similarly, z_t is the position in T of M_t (T_t). When z_s (z_t) has a -1 value, it means that the traversal reached a leaf and, in $maxval_s$ ($maxval_t$) the algorithm keeps the maximum value stored at that leaf node. Note that, T_s, T_t, $Lmax_s$, $Lmax_t$, and k are global variables.

In lines 1-11, the algorithm obtains the child of the processed node that contains the queried cell, provided that in a previous step, the algorithm did not reach a leaf node (signaled with z_s/z_t set to -1). In $maxval_s$ ($maxval_t$), the algorithm stores the maximum value stored in that node.

If the condition in line 12 is true, the algorithm has reached a leaf in both trees, and thus the values stored in $maxval_s$ and $maxval_t$ are added/subtracted to obtain the final result. If the condition of line 15 is true, the algorithm reaches a leaf in the snapshot. This is signaled by setting z_s to -1 and then a recursive call continues the process.

The *If* in line 19 treats the case of reaching a leaf in M_t. If the condition of line 20 is true, the algorithm uses bitmap eqB to check if the uniformity is in

Algorithm 1: getCell$(n, r, c, z_s, z_t, maxval_s, maxval_t)$ returns the value at cell (r, c)

```
if $z_s \neq -1$ then
    $z_s \leftarrow (rank_1(T_s, z_s) - 1) \cdot k^2 + 1$
    $z_s \leftarrow z_s + \lfloor r/(n/k) \rfloor \cdot k + \lfloor c/(n/k) \rfloor +1$
    $val_s \leftarrow Lmax_s[z_s - 1]$
    $maxval_s \leftarrow maxval_s - val_s$
end
if $z_t \neq -1$ then
    $z_t \leftarrow (rank_1(T_t, z_t) - 1) \cdot k^2+1$
    $z_t \leftarrow z_t + \lfloor r/(n/k) \rfloor \cdot k + \lfloor c/(n/k) \rfloor +1$
    $maxval_t \leftarrow Lmax_t[z_t - 1])$
end
if ($z_s > |T_s|$ or $z_s = -1$ or $T_s[z_s] = 0$) and ($z_t > |T_t|$ or $z_t = -1$ or $T_t[z_t] = 0$) then
  /* Both leafs */
    return $maxval_s + ZigZag\_Decoded(maxval_t)$
end
else if $z_s > |T_s|$ or $z_s = -1$ or $T_s[z_s] = 0$ then  /* Leaf in Snapshot */
    $z_s \leftarrow -1$
    return getCell($n/k, r \bmod (n/k), c \bmod (n/k), z_s, z_t, maxval_s, maxval_t$)
end
else if $z_t > |T_t|$ or $z_t = -1$ or $T_t[z_t] = 0$ then  /* Leaf in time instant */
    if $z_t \neq -1$ and $T_t[z_t] = 0$ then
        $eq \leftarrow eqB[rank_0(T_t, z_t)]$
        if $eq = 1$ then $z_t \leftarrow -1$ ;
        else return $maxval_s + ZigZag\_Decoded(maxval_t)$ ;
    end
    return getCell($n/k, r \bmod (n/k), c \bmod (n/k), z_s, z_t, maxval_s, maxval_t$)
end
else  /* Both internal nodes */
    return getCell($n/k, r \bmod (n/k), c \bmod (n/k), z_s, z_t, maxval_s, maxval_t$)
end
```

the original M_t submatrix or if it is in the submatrix resulting from applying the differences between the corresponding submatrix in M_s and M_t. A 1 in eqB implies the latter case, and this is solved by setting z_t to -1 and performing a recursive call. A 0 means that the treated original submatrix of M_t has the same value in all cells, and that value can be obtained adding/subtracting the values stored in $maxval_s$ and $maxval_t$, since the unique value in the submatrix of M_t is encoded as a difference with respect to the maximum value of the same submatrix of M_s, and thus the traversal ends.

The last case is that the treated nodes are not leaves, that simply requires a recursive call.

Retrieving cells with range of values in a time instant: $\langle [r_i, c_i] \rangle \leftarrow getCells(v_b, v_e, r_1, r_2, c_1, c_2, t)$ obtains from the raster of the time instant t, the positions of all cells within a region $[r_1, r_2] \times [c_1, c_2]$ containing values in the range $[v_b, v_e]$.

Again, if t is represented with a snapshot, the query is solved with the normal algorithm of the $\mathsf{k^2raster}$. Otherwise, as in the previous query, the search involves a synchronized top-down traversal of both trees. This time requires two main changes: (i) the traversal probably requires following several branches of both trees, since the queried region can overlap the submatrices corresponding to several nodes of the tree, (ii) at each level, the algorithm has to check whether the maximum and minimum values in those submatrices are compatible with the queried range, discarding those that fall outside the range of values sought.

5 Experimental evaluation

In this section we provide experimental results to show how $\mathsf{T}-\mathsf{k}^2\mathsf{raster}$ handles a dataset of raster data that evolve along time. We discuss both the space requirements of our representation and its performance at query time.

We used several synthetic and real datasets to test our representation, in order to show its capabilities. All the datasets are obtained from the TerraClimate collection [1], that contains high-resolution time series for different variables, including temperature, precipitations, wind speed, vapor pressure, etc. All the variables in this collection are taken in monthly snapshots, from 1958 to 2017. Each snapshot is a 4320×8640 grid storing values with $1/24°$ spatial resolution. From this collection we use data from two variables: TMAX (maximum temperature) is used to build two synthetic datasets, and VAP (vapor pressure) is compressed directly using our representation. Variable TMAX is a bad scenario for our approach, since most of the cells change their value between two snapshots. In this kind of dataset, our $\mathsf{T}-\mathsf{k}^2\mathsf{raster}$ would not be able to obtain good compression. Hence, we use TMAX to generate two synthetic datasets that simulate a slow, and approximately constant, change rate, between two real snapshots. We took the snapshots for January and February 2017 and built two synthetic datasets called T_100 and T_1000, simulating 100 and 1000 intermediate steps between both snapshots; however, note that to make comparisons easier we only take the first 100 time steps in both datasets. We also use a real dataset, VAP, that contains all the monthly snapshots of the variable VAP from 1998 to 2017. Note that, although we choose a rather small number of time instants in our experiments, the performance of our proposal is not affected by this value: it scales linearly in space with the number of time instants, and query times should be unaffected as long as the change rate is similar.

We compared our representation with two baseline implementations. The first, called $\mathsf{k}^2\mathsf{raster}$[6] is a representation that stores just a full snapshot for each time instant, without trying to take advantage of similarities between close time instants. The second baseline implementation, `NetCDF`, stores the different raster datasets in NetCDF format, using straightforward algorithms on top of the NetCDF library[7] (v.4.6.1) to implement the query operations. Note that NetCDF is a classical representation designed mainly to provide compression, through the use of Deflate compression over the data. Therefore, it is not designed to efficiently answer indexed queries.

We tested cell value queries (*getCellValue*) and range queries (*getCells*). We generated sets of 1000 random queries for each query type and configuration: 1000 random cell value queries per dataset, and sets of 1000 random range

[6] https://gitlab.lbd.org.es/fsilva/k2-raster
[7] https://www.unidata.ucar.edu/software/netcdf/

queries for different spatial window sizes (ranging from 4×4 windows to the whole matrix), and different ranges of values (considering cells with 1 to 4 possible values). To achieve accurate results, when the total query time for a query set was too small, we repeated the full query set a suitable number of times (in practice, 100 or 1000 times) and measured the average time per query.

All tests were run on an Intel (R) Core TM i7-3820 CPU @ 3.60GHz (4 cores) with 10MB of cache and 64GB of RAM, over Ubuntu 12.04.5 LTS with kernel 3.2.0-126 (64 bits). The code is compiled using gcc 4.7 with -O9 optimizations.

	$\mathsf{T-k^2raster}$ (varying t_δ)						$\mathsf{k^2raster}$	NetCDF (varying deflate level)			
	4	**6**	**8**	**10**	**20**	**50**		**0**	**2**	**5**	**9**
T_100	398.2	407.0	429.6	456.7	584.4	820.8	769.3	14241.3	615.3	539.5	528.0
T_1000	170.4	152.5	151.2	154.6	196.2	304.6	496.6	14241.3	435.0	344.7	323.6

Table 1. Space requirements (in MB) of $\mathsf{T-k^2raster}$, $\mathsf{k^2raster}$ and NetCDF over synthetic datasets.

Table 1 displays the space requirements for the datasets T_100 and T_1000 in all the representations. We tested our $\mathsf{T-k^2raster}$ with several sampling intervals t_δ, and also show the results for NetCDF using several deflate levels, from level 0 (no compression) to level 9. Our representation achieves the best compression results in both datasets, especially in T_1000, as expected, due to the slower change rate. In T_100, our approach achieves the best results for $t_\delta = 4$, since as the number of changes increases our differential approach becomes much less efficient. In T_1000, the best results are also obtained for a relatively small t_δ (6-8), but our proposal is still smaller than $\mathsf{k^2raster}$ for larger t_δ. NetCDF is only competitive when compression is applied, otherwise it requires roughly 20 times the space of our representations. In both datasets, NetCDF with compression enabled becomes smaller than the $\mathsf{k^2raster}$ representation, but $\mathsf{T-k^2raster}$ is able to obtain even smaller sizes.

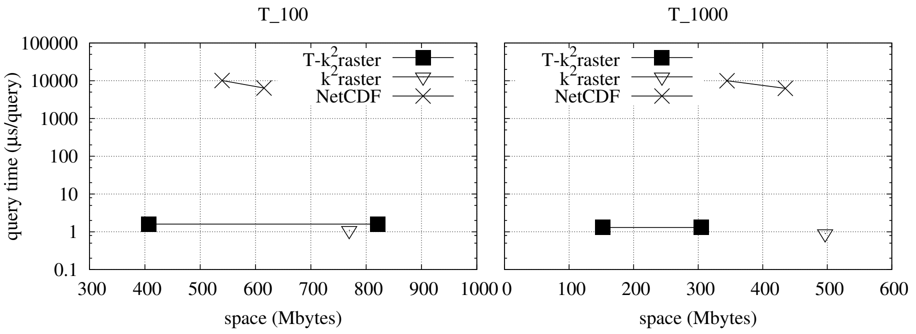


Fig. 3. Space/time trade-off on T_100 and T_1000 datasets for cell value queries.

Figure 3 shows the space/time trade-off for the datasets T_100 and T_1000 in cell value queries. We show the results only for NetCDF with compression enabled (deflate level 2 and 5), and for $\mathsf{T-k^2raster}$ with a sampling interval of 6 and 50. The $\mathsf{T-k^2raster}$ is slower than the baseline $\mathsf{k^2raster}$, but is much smaller if a good t_δ is selected. Note that we use two extreme sampling intervals to show the consistency of query times, since in practice only the best approach in space would be used for a given dataset. In our experiments we work with a fixed t_δ, but straightforward heuristics could be used to obtain an space-efficient T−k^{2}raster without probing for different periods: for instance, the number of nodes in the tree of differences and in the snapshot is known during construction, so a new snapshot can be built whenever the size of the tree of differences increases above a given threshold.

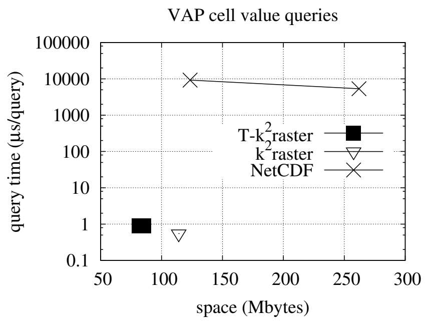
		T_100					T_1000				
		T−k^{2}raster		k^{2}raster	NetCDF		T−k^{2}raster		k^{2}raster	NetCDF	
wnd	**rng**	**6**	**50**		**2**	**5**	**6**	**50**		**2**	**5**
16	**1**	3.6	3.8	2.8	6130	10070	3.3	3.4	2.5	6160	10020
	4	5.1	5.5	3.6	6240	10100	3.5	3.5	2.6	6160	10100
256	**1**	222.9	248.1	163.9	9610	15330	207.1	228.9	167.6	9370	15110
	4	429.3	489.4	301.7	9340	14790	213.4	234.3	172.7	9510	15240
ALL	**1**	111450	126220	78250	443830	580660	79650	89380	63350	436400	568730

Table 2. Range query times over T_100 and T_1000 datasets. Times shown in μs/query for different spatial windows (wnd) and range of values (rng).

Table 2 shows an extract of the range query times for all the representations in datasets T_100 and T_1000. We only include here the results for T−k^{2}raster with a t_δ of 6 and 50, and for NetCDF with deflate level 2 and 5, since query times with the other parameters report similar conclusions. We also show the results for some relevant spatial window sizes and ranges of values. In all the cases, T−k^{2}raster is around 50% slower than k^{2}raster, due to the need of querying two trees to obtain the results. However, the much smaller space requirements of our representation compensate for this query time overhead, especially in T_1000. NetCDF, that is not designed for this kind of queries, cannot take advantage of spatial windows or ranges of values, so it is several orders of magnitude slower than the other approaches. The last query set (ALL) involves retrieving all the cells in the raster that have a given value (i.e. the spatial window covers the complete raster). In this context, NetCDF must traverse and decompress the whole raster, but our representation cannot take advantage of its spatial indexing capabilities, so this provides a fairer comparison. Nevertheless, both T−k^{2}raster and k^{2}raster are still several times faster than NetCDF in this case, and our proposal remains very close in query times to the k^{2}raster baseline.

Figure 4 (left) shows the space/time trade-off for the real dataset VAP. Results are similar to those obtained for the previous datasets: our representation, T−k^{2}raster, is a bit slower in cell value queries than k^{2}raster, but also requires significantly less space. The NetCDF baseline is much slower, even if it becomes competitive in space when deflate compression is applied.

Finally, Figure 4 (right) displays the query times for all the alternatives in range queries over the VAP dataset. The k^{2}raster is again a bit faster than the T−k^{2}raster, as expected, but the time overhead is within 50%. NetCDF is much slower, especially in queries involving small windows, as it has to traverse and decompress a large part of the dataset just to retrieve the values in the window.



		$\mathsf{T-k^2raster}$		$\mathsf{k^2raster}$	`NetCDF`	
wnd	**rng**	**6**	**50**		**2**	**5**
4	**1**	2.2	2.2	1.6	5570	9520
	2	2.0	2.0	1.4	5530	9430
	3	1.9	1.8	1.3	5580	9430
	4	1.7	1.7	1.3	5550	9470
16	**1**	4.3	4.2	3.1	5670	9670
	2	3.7	3.7	2.7	5630	9730
	3	3.3	3.4	2.4	5660	9660
	4	2.9	3.0	2.2	5720	9740
64	**1**	26.4	26.1	19.5	6150	10470
	2	21.1	21.6	16.1	6140	10440
	3	16.8	17.3	12.9	6130	10450
	4	16.2	16.6	12.6	6220	10660
256	**1**	239.6	242.5	179.4	8720	14820
	2	207.2	218.7	161.0	8660	14640
	3	181.7	187.9	140.3	8590	14430
	4	142.2	146.5	112.9	8300	14020
ALL	**1**	60400	62900	46200	411700	552500

Fig. 4. Results for VAP dataset. Left plot shows space/time tradeoff for cell value queries. Right table shows query times for range queries. Times in μs/query.

Note that even if the window covers the complete raster, $\mathsf{T-k^2raster}$ and $\mathsf{k^2raster}$ achieve significantly better query times.

6 Conclusions and future work

In this work we introduce a new representation for time-evolving raster data. Our representation, called $\mathsf{T-k^2raster}$, is based on a compact data structure for raster data, the $\mathsf{k^2raster}$, that we extend to efficiently manage time series. Our proposal takes advantage of similarities between consecutive snapshots in the series, so it is especially efficient in datasets where few changes occur between consecutive time instants. The $\mathsf{T-k^2raster}$ provides spatial and temporal indexing capabilities, and is also able to efficiently filter cells by value. Results show that, in datasets where the number of changes is relatively small, our representation can compress the raster and answer queries very efficiently. Even if its space efficiency depends on the dataset change rate, the $\mathsf{T-k^2raster}$ is a good alternative to compress raster data with high temporal resolution, or slowly-changing datasets, in small space.

As future work, we plan to apply to our representation some improvements that have already been proposed for the $\mathsf{k^2raster}$, such as the use of specific compression techniques in the last level of the tree. We also plan to develop an adaptive construction algorithm, that selects an optimal, or near-optimal, distribution of snapshots to maximize compression.

References

1. Abatzoglou, J.T., Dobrowski, S.Z., Parks, S.A., Hegewisch, K.C.: Terraclimate, a high-resolution global dataset of monthly climate and climatic water balance from 1958–2015. Scientific Data **5**(170191) (2017)
2. de Bernardo, G., Álvarez-García, S., Brisaboa, N.R., Navarro, G., Pedreira, O.: Compact Querieable Representations of Raster Data. In: Proceedings of the International Symposium on String Processing and Information Retrieval (SPIRE). pp. 96–108 (2013)
3. Botea, V., Mallett, D., Nascimento, M.A., Sander, J.: Pist: An efficient and practical indexing technique for historical spatio-temporal point data. GeoInformatica **12**(2), 143–168 (2008)
4. Brisaboa, N.R., Ladra, S., Navarro, G.: DACs: Bringing direct access to variable-length codes. Information Processing and Management **49**(1), 392–404 (2013)
5. Brisaboa, N.R., Ladra, S., Navarro, G.: Compact representation of web graphs with extended functionality. Information Systems **39**(1), 152–174 (2014)
6. Couclelis, H.: People manipulate objects (but cultivate fields): Beyond the raster-vector debate in GIS. In: Proceedings of GIS: from space to territory - theories and methods of spatio-temporal reasoning. pp. 65–77 (1992)
7. Jacobson, G.: Succinct static data structures. Ph.D. thesis, Carnegie-Mellon (1988)
8. Ladra, S., Paramá, J.R., Silva-Coira, F.: Scalable and queryable compressed storage structure for raster data. Information Systems (72), 179–204 (2017)
9. Mennis, J., Viger, R., Tomlin, C.D.: Cubic map algebra functions for spatio-temporal analysis. Cartography and Geographic Information Science **32**(1), 17–32 (2005)
10. Nascimento, M.A., Silva, J.R.O.: Towards historical R-trees. In: Proceedings of the 1998 ACM symposium on Applied Computing, SAC'98. pp. 235–240. ACM (1998)
11. Navarro, G.: Compact Data Structures – A practical approach. Cambridge University Press (2016)
12. Pinto, A., Seco, D., Gutiérrez, G.: Improved queryable representations of rasters. In: Proceedings of the 2017 Data Compression Conference (DCC). pp. 320–329 (2017)
13. Tao, Y., Papadias, D.: MV3R-tree: A spatio-temporal access method for timestamp and interval queries. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB). pp. 431–440 (2001)
14. Vazirgiannis, M., Theodoridis, Y., Sellis, T.K.: Spatio-temporal composition and indexing for large multimedia applications. ACM Multimedia Systems Journal **6**(4), 284–298 (1998)