FastMosaic in Action: A New Mosaic Operator for Array DBMSs

Ramon Antonio Rodriges Zalipynis
HSE University
Moscow, Russia
rodriges@gis.land

ABSTRACT

Array DBMSs operate on N-d arrays. During the Data Ingestion phase, the widely used MOSAIC operator ingests a massive collection of overlapping arrays into a single large array, called mosaic. The operator can utilize sophisticated statistical and machine learning techniques, e.g. Canonical Correlation Analysis (CCA), to produce a high quality *seamless mosaic* where the contrasts between the values of cells taken from input overlapping arrays are minimized. However, the performance bottleneck becomes a major challenge when applying such advanced techniques over increasingly growing array volumes. We introduce a new, scalable way to perform CCA that is orders of magnitude faster than the popular Python's scikit-learn library for the purpose of array mosaicking. Furthermore, we developed a hybrid web-desktop application to showcase our novel FASTMOSAIC operator, based on this new CCA. A rich GUI enables users to comprehensively investigate in/out arrays, interactively guides through an end-to-end mosaic construction on real-world geospatial arrays using FASTMOSAIC, facilitating a convenient exploration of the FASTMOSAIC pipeline and its internals.

PVLDB Reference Format:
Ramon Antonio Rodriges Zalipynis. FastMosaic in Action: A New Mosaic Operator for Array DBMSs. PVLDB, 16(12): 3938 - 3941, 2023.
doi:10.14778/3611540.3611590

1 INTRODUCTION

Array DBMSs are becoming increasingly comprehensive systems, enabling N-d array storage, processing, and even visualization [10].

The *Data Management Problem* we tackle is the improvement of the speed and quality of the Data Ingestion phase in Array DBMSs, a mandatory data life cycle stage for all arrays that arrive under the control of Array DBMSs. Specifically, we focus on the well-known MOSAIC operator of Array DBMSs. For example, ORACLE SPATIAL provides the `SDO_GEOR_AGGR.mosaicSubset` procedure [7], CHRONOSDB has the MOSAIC command [9], and RASDAMAN is equipped with the MOSAIC recipe [6].

N-d arrays are natural models for many important data types [1], of which Earth remote sensing data constitute one of the most significant Array DBMS workloads [6, 7, 9, 10]. Hence, consider a practical example related to real-world geospatial arrays. A single array, e.g. a satellite scene, often does not fully cover the area of interest. In this case, the MOSAIC operator, equipped by *Data Science* techniques, is used to fuse a large collection of input arrays into a single seamless mosaic (a large N-d array where the visibility of stitches between individual input arrays is minimized; visual quality evaluation prevails, no numerical metrics are routinely applied [4]).

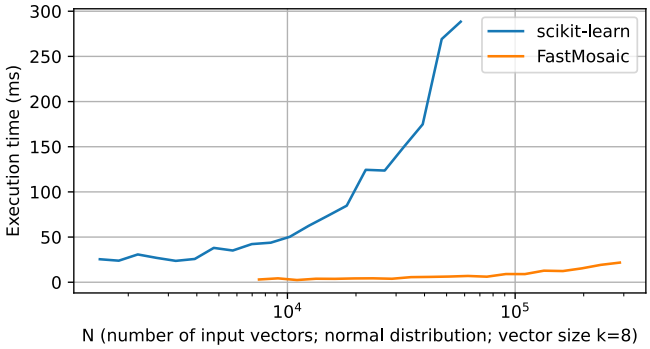


Figure 1: CCA: FASTMOSAIC VS. Python's "scikit-learn" [8]

Hence, MOSAIC is an essential Data Ingestion step that helps to solve vital daily tasks like urban planning, agriculture monitoring, forestry control, and rapid-response in disaster management [1].

To produce high quality mosaics, algorithms resort to advanced techniques. For example, IR-MAD relies on Canonical Correlation Analysis (CCA) [4], a popular method for finding correlations in multidimensional datasets [2]. CCA is widely used in Data Science for dimensionality reduction and discovering latent variables.

The major challenge in using CCA or other advanced approaches comes from rapidly growing data volumes (array sizes) that require ever more scalable algorithms. For instance, CCA must consider billions of array cells for an averagely-sized mosaic [4]. However, available CCA implementations like [8] are compute-intensive and do not scale with contemporary array data volumes, fig. 1.

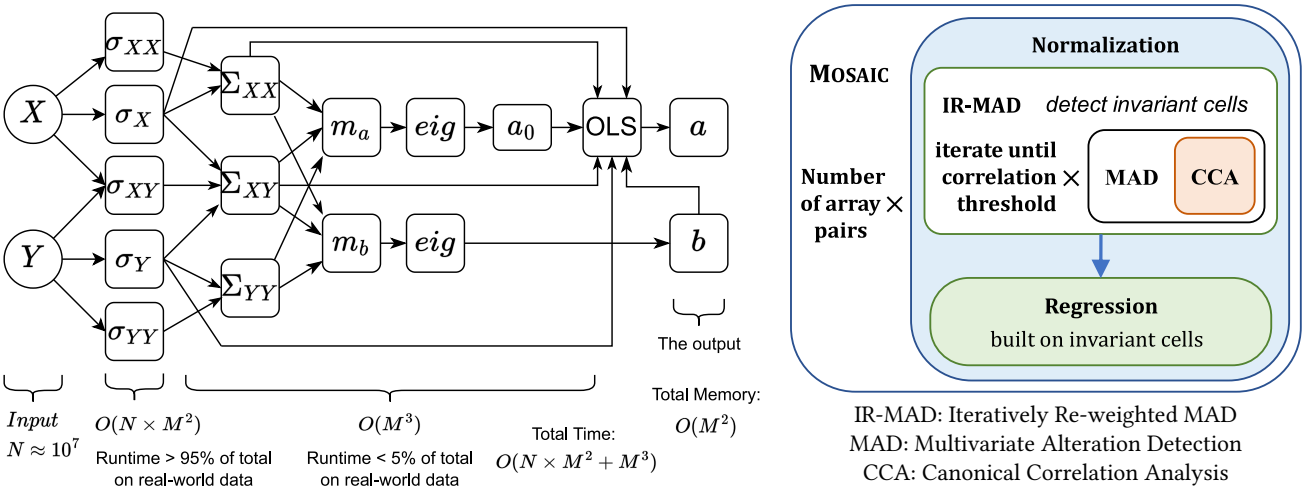
Recent surveys indicate that little techniques focus on the mosaicking scalability, so the speed is set out as a key challenge and improvement aspect of next generation mosaicking schemes [4, 5].

To date, Array DBMSs support basic or no value normalization in array mosaicking. ORACLE SPATIAL provides Linear Stretching, Statistics Matching, and Histogram Matching options [7]. CHRONOSDB does not perform color correction for the MOSAIC operation [9], similarly to RASDAMAN [6]. Due to limited color or value correction algorithms, users tend to ingest mosaics into Array DBMSs created outside. To avoid costly data movements, Array DBMSs must directly support high-quality and scalable mosaic algorithms.

Novelty and Contributions.

(1) We introduce a new, scalable way to perform Canonical Correlation Analysis (CCA), a popular Data Science technique, in linear time in the context of the Data Ingestion phase in Array DBMSs (the MOSAIC operator).

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611590



(a) **CCA Compute Graph: Proposed Approach**

(b) **FastMosaic Structure**

Figure 2: Compute Graph of the Proposed CCA Algorithm and FastMosaic Structure

(2) We showcase FastMosaic, a new Array DBMS mosaic operator, equipped with our CCA algorithm which is orders of magnitude faster for the purpose of array mosaicking compared to the popular Python implementation, fig. 1.
(3) We built a web-desktop application to engage interested users in a vivid inspection of FastMosaic via an end-to-end mosaic creation on real-world data in a rich GUI.

2 FASTMOSAIC: AN ARRAY DBMS OPERATOR

FastMosaic improves IR-MAD, because it runs with no or little manual interaction: an important property for Big Data applications. In addition, it can generate high quality array mosaics, even for very complex value distributions of overlapping input array cells [4].

The FastMosaic structure appears in fig. 2b. This section provides core theoretical foundations of FastMosaic. Section 3 describes an end-to-end mosaic construction on real-world geospatial arrays, complementing the FastMosaic description.

As input, FastMosaic accepts a set of 3-d overlapping arrays shaped $lat \times long \times k$, where k is the number of bands. Typically, $lat \approx long \approx 8000$ and $k \approx 10$. For a pair of input arrays (reference and subject), we estimate the probability $P(no\ change)$ for their overlapping cells using CCA. Invariant cells should have $P(no\ change) > 0.95$. Next, the relative normalization is performed: on the invariant cells an orthogonal regression is built, whose k pairs of coefficients are applied to all cells of the subject array. The resulting array is merged with the reference array, this larger array replaces the array pair in the input set, and the procedure is repeated until there is only 1 array left: the resulting array mosaic.

2.1 Collecting Statistics

At this early stage, to speedup subsequent stages, we collect statistics that will serve as "building blocks" for the main formulae.

We treat N cell pairs of two overlapping arrays as a pair of random variables X and Y of dimension k: $X_i = \{X_{i,1}, X_{i,2}, \ldots, X_{i,N}\}$, where $X_{i,j}$ is the value of cell $j \in [1, N]$ from band $i \in [1, k]$. We define Y in the same way. We treat the output array of weights as a 1-d vector w of size N. We let $w_i = 1$ before the first iteration $\forall\ i$: $w = \{w_1, w_2, ..., w_N\}$.

Now, we present the statistics formulae and the calculation of covariance matrices for X (the formulae for Y are analogues). First, we compute $\sigma_{X,i} = \sum_j^N X_{i,j} w_j$, the weighted sum of cells from band i: $\sigma_X = \{\sigma_{X_1}, \sigma_{X_2}, \ldots, \sigma_{X_k}\}$.

Then, we compute σ_{XY}, a matrix of weighted sums of products X and Y: $\sigma_{XY} = X_i^T (Y_i \odot w)$, $\sigma_{XX} = X_i^T (X_i \odot w)$, $\sigma_{YY} = Y_i^T (Y_i \odot w)$, where $\odot$ is a cell-wise product.

In addition, we compute Σ_{XY}, a matrix of a weighted covariance of variables X and Y:

$$\Sigma_{XY} = \frac{\sigma_{XY}}{\sum w - 1} - \frac{\sigma_X \sigma_Y^T}{\sum w (\sum w - 1)} \tag{1}$$

Similarly, Σ_{XX} and Σ_{YY} are computed as

$$\Sigma_{XX} = \frac{\sigma_{XX}}{\sum w - 1} - \frac{\sigma_X \sigma_X^T}{\sum w (\sum w - 1)} \tag{2}$$

$$\Sigma_{YY} = \frac{\sigma_{YY}}{\sum w - 1} - \frac{\sigma_Y \sigma_Y^T}{\sum w (\sum w - 1)} \tag{3}$$

In the compute graph, the collection of statistics takes over 95% of the CCA runtime, fig. 2a. Other approaches do not work in one pass over the input data, so their iterations are very expensive.

2.2 Proposed Linear Algorithm for CCA

We now present our linear-time formulae to compute CCA, fig. 2a. We designed the formulae such that it is possible to use the previously collected statistics (section 2.1) and perform the calculations of both CCA canonical variables (U and V) and the normalization coefficients (β_i and ϵ_i) in a single pass over the input data.

Recall that CCA maximizes the correlation $corr(a^T X, b^T Y)$ by seeking coefficients a and b. Random variables $U = a^T X, V = b^T Y$ are called a pair of canonical variables. We can find coefficients a, b by decomposing the covariance matrix on eigenvectors and

eigenvalues. Then, we can get a pair of coefficients a_i, b_i from each eigenvector. Eigenvectors that correspond to the largest eigenvalues will correspond to the largest values of $corr(a_i^T X, b_i^T Y)$.

When computing CCA, we perform the decomposition of covariance matrices on eigenvectors. We compute matrices m_a and m_b as follows:

$$m_a = \Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^{T}\Sigma_{XX}^{-1/2} \tag{4}$$

$$m_b = \Sigma_{YY}^{-1/2}\Sigma_{XY}^{T}\Sigma_{XX}^{-1}\Sigma_{XY}\Sigma_{YY}^{-1/2} \tag{5}$$

Then, we compute vectors a_0 and b as eigenvectors of matrices m_a and m_b (4), (5):

$$a_0 = eigvectors(m_a)\Sigma_{XX}^{-1/2} \tag{6}$$

$$b = eigvectors(m_b)\Sigma_{YY}^{-1/2} \tag{7}$$

Now, consider the following observation. The resulting vectors a_0 and b are proportional to true vectors: $a_{true} = \zeta a$, $b_{true} = \omega b$. However, for the correct functioning of the mosaic operator, we have to find overlapping array cells whose value differences cannot be explained by linear dependencies. In other words, we have to find array cells whose canonical variables do not correlate. If we find coefficients a, b that look like $a = \gamma a_{true}, b = \gamma b_{true}$, then array cells with the largest values of $(a^T X - b^T Y)^2$ will be the cells we are looking for. The coefficient γ will be canceled during the subsequent normalization of $U_i - V_i$. To achieve this effect, we can multiply one of the variables on the correction factor:

$$a = \beta a_0$$

We can obtain the coefficient β using a linear regression on the variables U and V:

$$a = \beta a_0 = (U^T W U)^{-1} U^T W V a_0 \tag{8}$$

where

$$W = diagonal[\{w_1, w_2, ..., w_k\}]$$

As long as our algorithm does not keep U and V in the operating memory during the computation of CCA, we can express the linear regression using the **previously collected statistics**, section 2.1:

$$\begin{aligned} U^T W U &= \begin{bmatrix} \sum w_i & \sum w_i u_i \\ \sum w_i u_i & \sum w_i u_i^2 \end{bmatrix} \\ &= \begin{bmatrix} \sum w_i & a^T \sigma_X \\ a^T \sigma_X & grandsum[(aa^T) \odot \Sigma_{XX}] \end{bmatrix} \end{aligned} \tag{9}$$

$$U^T W V = \begin{bmatrix} \sum w_i v_i \\ \sum w_i u_i v_i \end{bmatrix} = \begin{bmatrix} b^T \sigma_Y \\ grandsum[(ab^T) \odot \Sigma_{XY}] \end{bmatrix} \tag{10}$$

As an alternative, it is also possible to implement a "regularized" CCA by summing up the covariance matrices with an identity matrix, multiplied by the regularization coefficient.

2.3 Constructing the Statistical Test

For each cell of the resulting mosaic array, the probability of the cell invariability (stationarity) is estimated with the χ^2 distribution:

$$P(no\ change) = \chi^2_{cdf}\left(\sum M_i^2\right) \tag{11}$$

where $M = \{M_1, M_2, ..., M_k\}$ is a vector of k standardly distributed random variables, obtained from the normalized difference of k pairs of canonical variables:

$$M_i = \frac{(U_i - V_i) - \overline{M_i}}{std(M_i)} \tag{12}$$

Every member in eq. (12) can be expressed using the **previously collected statistics** (section 2.1), and vectors a, b, computed by **our linear CCA algorithm**, section 2.2: $\overline{M_i} = \overline{U_i} - \overline{V_i}$ such that

$$\overline{U_i} = a^T \frac{\sigma_X}{\sum w} \text{ and } \overline{V_i} = b^T \frac{\sigma_Y}{\sum w}$$

Furthermore, $UV_{cov} = diag(a^T \Sigma_{XY} b)$ such that

$$U_{var} = diag(a^T \Sigma_{XX} a) \text{ and } V_{var} = diag(b^T \Sigma_{YY} b)$$

Now we can find cells $P(no\ change) > \Theta \in [0.95, 0.99]$ and build orthogonal regression only on these invariant cells: $Y_i = \beta_i X_i + \epsilon_i$. Note that transformations superimpose during the mosaic construction, leading to a non-linear transformation of input arrays.

3 FASTMOSAIC IN ACTION

3.1 Illustrative Dataset

We prepared a dataset using real-world geospatial arrays: Landsat 8 satellite scenes. Landsat is one of the most popular and the longest continuous space-based record of Earth's land [3]. We selected an interesting agricultural area featuring central pivot irrigation fields (Saudi Arabia): a set of 3×4 scenes, bands 1−7, paths 166−169, rows 41 − 43, GeoTIFF, fig. 3. We deliberately selected scenes acquired on the 22 of February and 15, 16, 17 of March and 2023 (different dates) to explore diverse conditions for the mosaicking algorithm.

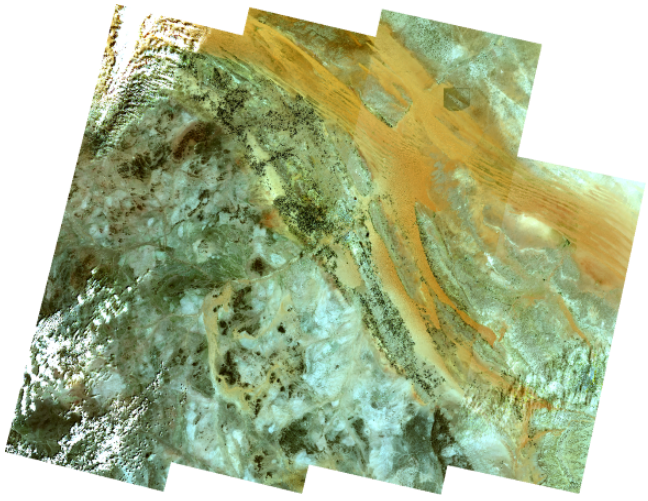


Figure 3: A collection of input arrays (natural colors)

3.2 FastMosaic Interactive GUI

The application guides users through the end-to-end workflow (fig. 4) of constructing a seamless mosaic (fig. 5) on raw input arrays (fig. 3) via an interactive and rich GUI: see below.

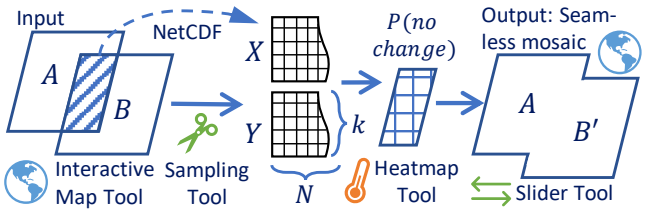


Figure 4: FastMosaic Workflow Overview (2 input arrays)

Two mosaicking modes are available, both resulting in a mosaic: (1) manual plan, batch run on all input arrays, and (2) step-by-step, two input arrays. Mode №1 (breadth-first) invites playing with the impact caused by the order of adding arrays to the overall, final mosaic. On the contrary, Mode №2 (depth-first) enables an in-depth investigation of the FastMosaic workflow on an array pair, fig. 4.

Interactive Map Tool makes it possible to explore raw input arrays, intermediate and final mosaicking results for both modes. The user starts by examining the Interactive Map with the input arrays: overlapping areas have varying contrasts and rich contents (clouds, sand, urban settlements, agricultural fields), section 3.1.

Mosaic Plan Tool (Mode №1). The user interactively builds a tree (mosaic execution plan) by drawing arrows to connect array pairs (snapping supported) and set the fusion order, fig. 6. At step №i, the array at the start of arrow №i joins the mosaic built so far.

Slider Tool is useful for comparing the input and the resulting mosaic for an overlapping area (both modes). The user can select a rectangular region and collate the input to the left vs. the output to the right of the slider (the blue circle; draggable left↔right), fig. 5.

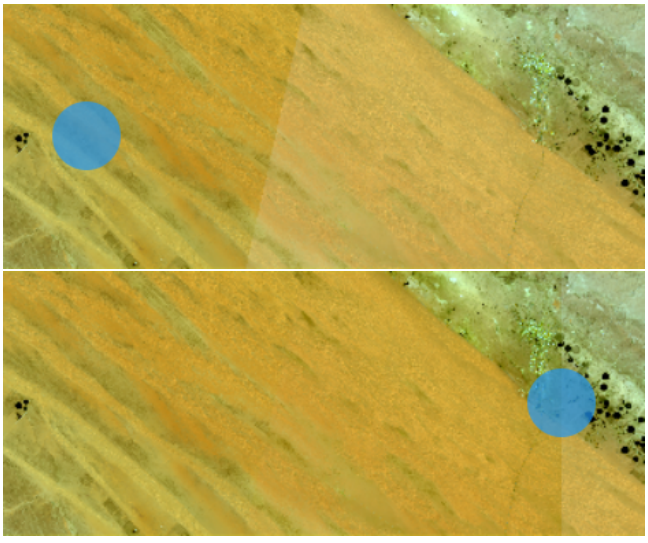


Figure 5: Slider Tool in Action: Collating Input and Output

In Mode №2, the user selects a pair of overlapping 3-d arrays and creates X, Y arrays with the **Sampling Tool**: the CCA input. Then, the user sets the FastMosaic options to generate $P(no\ change)$, section 2.3: (1) CCA implementation: this paper or Python, fig. 1, (2) correlation threshold (the significance of the change in correlations $corr(U, V)$), or (3) the max number of iterations, fig. 2b. The user will assess the FastMosaic outputs ($P(no\ change)$ and the resulting mosaic) on any iteration number by tuning the last parameter and see that choice (1) does not affect the mosaic quality.

Correlation Plot Tool. During the CCA execution, the user can visually track the convergence of the algorithm with the highly interactive plot updated at each iteration step, fig. 6. The line number i plots the correlation of canonical variables $corr(U_i, V_i)$, fig. 6.

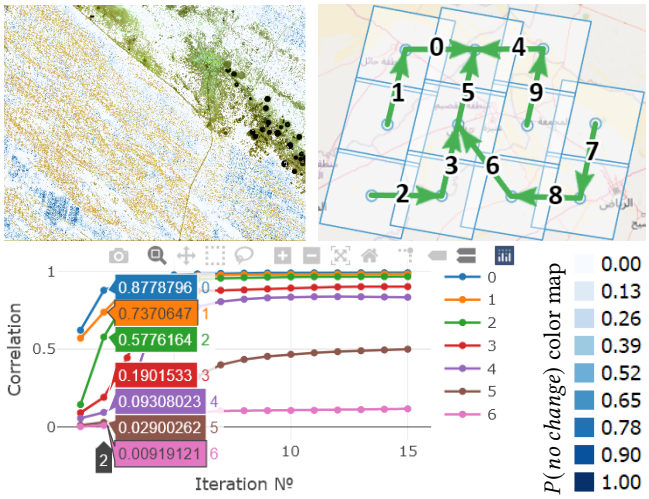


Figure 6: $P(no\ change)$ Heatmap (for fig. 5), Mosaic Execution Plan, Interactive Plot (Correlations of Canonical Variables)

Heatmap Tool plots $P(no\ change)$, so the user can thoroughly inspect cells in the overlapping area that are likely invariant, fig. 6.

Finally, the user provides a $P(no\ change)$ threshold in $[0.95, 0.99]$ to compute the regression coefficients on the invariant cells, build the final mosaic, and study it with the **Map** and **Slider** Tools, fig. 4.

Paper homepage: https://wikience.github.io/fastmosaic2023

REFERENCES

[1] ArcGIS Book 2023. https://learn.arcgis.com/en/arcgis-imagery-book/.
[2] Harold Hotelling. 1936. Relations between two sets of variates. *Biometrika* 28, 3/4 (1936), 321–377.
[3] Landsat. 2023. https://www.usgs.gov/landsat-missions.
[4] Xinghua Li et al. 2019. Remote sensing image mosaicking: Achievements and challenges. *IEEE Geoscience and Remote Sensing Magazine* 7, 4 (2019), 8–22.
[5] Bose Alex Lungisani et al. 2022. The Current State on Usage of Image Mosaic Algorithms. *Scientific African* (2022), e01419.
[6] RasDaMan Mosaic. 2023. https://doc.rasdaman.org/05_geo-services-guide.html#data-import-recipe-mosaic-map.
[7] Oracle Database Release 21c. https://docs.oracle.com/en/database/oracle/oracle-database/21/georsimage-processing-virtual-mosaic.html.
[8] Python scikit-learn: sklearn.cross_decomposition.CCA 2023. http://scikit-learn.org/stable/modules/generated/sklearn.cross_decomposition.CCA.html.
[9] Ramon Antonio Rodriges Zalipynis. 2018. ChronosDB: Distributed, File Based, Geospatial Array DBMS. *PVLDB* 11, 10 (2018), 1247–1261.
[10] Ramon Antonio Rodriges Zalipynis. 2021. Array DBMS: Past, Present, and (Near) Future. *PVLDB* 14, 12 (2021), 3186–3189.