Exp Astron (2013) 35:245–282
DOI 10.1007/s10686-012-9295-0

ORIGINAL ARTICLE

Sub-image data processing in Astro-WISE

Johnson Mwebaze · Danny Boxhoorn · John McFarland · Edwin A. Valentijn

Received: 11 June 2011 / Accepted: 21 February 2012 / Published online: 28 March 2012

Abstract Most often, astronomers are interested in a source (e.g., moving, variable, or extreme in some colour index) that lies on a few pixels of an image. However, the classical approach in astronomical data processing is the processing of the entire image or set of images even when the sole source of interest may exist on only a few pixels of one or a few images. This is because pipelines have been written and designed for instruments with fixed detector properties (e.g., image size, calibration frames, overscan regions, etc.). Furthermore, all metadata and processing parameters are based on an instrument or a detector. Accordingly, out of many thousands of images for a survey, this can lead to unnecessary processing of data that is both time-consuming and wasteful. We describe the architecture and an implementation of sub-image processing in Astro-WISE. The architecture enables a user to select, retrieve and process only the relevant pixels in an image where the source exists. We show that lineage data collected during the processing and analysis of datasets can be reused to perform selective reprocessing (at sub-image level) on datasets while the remainder of the dataset is untouched, a difficult process to automate without lineage.

Keywords Data lineage · Provenance · Astro-WISE · Sub-image processing

J. Mwebaze (✉) · D. Boxhoorn · J. McFarland · E. A. Valentijn
Kapteyn Astronomical Institute,
University of Groningen, Landleven 12,
9700 AV Groningen, The Netherlands
e-mail: jmwebaze@cit.ac.ug

1 Introduction

The nature of astronomical (or scientific) data processing is changing. Datasets are doubling in size and number every year, with archives growing beyond petabyte scales [26]. Similarly, the processing is increasing in complexity requiring laboriously sophisticated techniques and expertise in both data reduction and analysis. Often such processing involves evaluation of a variety of processing techniques and different datasets, tweaking parameters, verifying data quality, repeating data derivation and customization of the results. Consequently more data is generated as new results are derived. It is therefore crucial for the experimenter or other users of the system to understand how a result was derived, what data was used, and how it was selected.

Astronomical systems must also provide scalable (the ability to handle growing amounts of data/processes in a graceful manner) processing solutions, and allow seamless access to various distributed resources and archives to facilitate such data analysis. But, how will users retrieve and process data they are interested in? For moving an entire dataset, a surprisingly low tech approach is often used. However, most clients have neither the time nor resources to effectively store and analyse such huge datasets. Moreover, the cost of data transfer is not substantially cheaper today than the price of disks to store the data. Therefore, transferring a minimal set of data is critical. Also, the traditional approach of simply moving the data to where needed is inherently not scalable primarily due to network-related and client processing constraints.

We use the above views to motivate the need to trace lineage and also the need to use lineage data to enable the effective and selective reuse of data, knowledge and experiences from previous experiments to aid both expert and non-expert distributed users in performing scientific analysis. The lineage of a data item consists of its entire processing history. This includes its origin (e.g., the identifier of the base data set, the recording instrument, the instrument's operating parameters) as well as all subsequent processing steps (algorithms and respective parameters) applied to it.

Data lineage (provenance) is a well-defined problem with known solutions as pointed out in recent workshops [20] and surveys [4, 8, 11, 19]. The use of data lineage has also gained significant attention [7, 25]. Several workflow management systems do exploit data lineage for different purposes. To the best of our knowledge, this is the first work that leverages lineage information to support sub-image processing to simplify and automate the reprocessing of objects. Since we are working with pixels, this framework required lineage at pixel level. We extended our lineage model presented in [21] to trace lineage at pixel level and then use pixel lineage for sub-image processing.

The framework for sub-image processing we have designed and implemented, enables users to select, retrieve and process only relevant pixels on an image where the source exists. However some image operations may not be carried out at sub-image level. Moreover all processing parameters and metadata are based on fixed detector properties. We claim that such a framework is possible if we can retrieve an entire back trace of the processing

history of an image at pixel level, and then be able to perform operations that are otherwise difficult (or impossible) to execute at sub-image level.

The goal of this paper is twofold; firstly, we extend our lineage model presented in [21] to trace lineage at pixel level and we show that it accurately locates all pixels that are involved in the processing of a result. Secondly, we show that using lineage we are able to support sub-image processing. We specifically focus on Astro-WISE, however, our discussion of the requirements of sub-image processing can be applied to the general category of all observational sciences.

The rest of the paper is organized as follows: Section 2 briefly provides an overview of lineage tracing in Astro-WISE. In Section 3, we review astronomical data and transformations related to this work. We present the pixel lineage framework in Section 4, and in Section 5, we describe and implement the sub-image processing framework. In Section 6, we describe the management of image cutouts and how awe keeps track of dependencies during si processing. We present a use case to demonstrate the effectiveness of sub-image processing in Section 7. We review related work in Section 8 and we present our conclusions in Section 9.

2 Lineage in Astro-WISE

Astro-WISE[1] [3, 17] is a distributed system for processing, analysing and disseminating wide field imaging data. Astro-WISE enables astronomers to perform scientific experiments in a distributed environment. The full description of lineage tracing in Astro-WISE is described in [21]. Astro-WISE tracks all parameters used (or created) during runtime that induces any dependency relationship between objects, i.e., parameters that would affect the state of an object. These relationships are the links that allow Astro-WISE to deduce the lineage of an object.

A request for any object in Astro-WISE, automatically invokes the process of generating a Directed Acyclic Graph (DAG). A directed acyclic graph is formed by a collection of vertices and directed edges with each edge connecting one vertex to another. An example of a DAG is shown in Fig. 1. The algorithm for creating the DAG of an object is to first find which derivation contains the object as output, then for each input of the associated transformation find the derivation that contains it as output, iterating until all the dependencies are resolved. The rectangle nodes in Fig. 1 represent the modules used during processing and edges represent data flow between the modules. The circle nodes represent data inputs/output of a module. Each execution of an experiment may vary the parameters (diamond nodes) to the modules in the specification which might result in different outputs. The dataflow pattern (or DAG) in Fig. 1 captures the topology of a typical experiment presented

[1] http://www.astro-wise.org

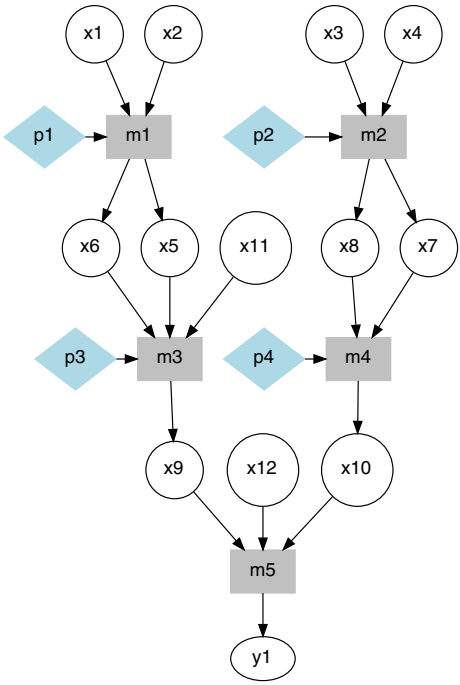


Fig. 1 DAG: an example of a data lineage graph

and therefore is used to represent a lineage graph of an item processed in Astro-WISE.

3 Astronomical transformations

This granularity of lineage traced in Astro-WISE as described in Section 2 does not include transformations at pixel level. For sub-image processing, we need to trace lineage at pixel level. In this section we discuss the properties of astronomical images and astronomical image transformations, which form the basis of pixel lineage tracing.

3.1 Astronomical data

Images of the sky and catalogue data are perhaps the most important astronomical data products. Most of the catalogue data contains astronomical[2] (celestial) sources which are ultimately derived from images. The detection of sources from an image consists of identifying and separating image regions which have different properties (e.g., brightness, colour, texture). Therefore an

[2] Any of the natural objects that can be seen in our sky, including stars, planets, moons, asteroids, galaxies, and comets.

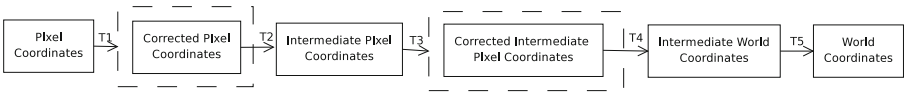


Fig. 2 Flow chart showing a conversion of pixel coordinates to world-coordinates showing distortion corrections enclosed in the *dashed boxes* [16]

astronomical source is a group of connected pixels that exceed some threshold selected through some detection process and for which the flux contribution of the source is believed to be dominant over that of other objects.

The de facto standard format for astronomical images is the Flexible Image Transfer System (FITS).[3] Besides pixel data, FITS images usually contain essential information such as the celestial coordinates used and time of observation (if applicable) as well as other information in the headers. Each element in the image data array is mapped onto a sky/world coordinate on the sky or within a spectrum. In this paper, sky and world coordinates are used interchangeably.

The FITS coordinate system should be invertible in the sense that a pixel coordinate, when transformed to a world coordinate, must be uniquely recoverable from that world coordinate. In practical terms, it means that two or more different pixel coordinates should not map to the same world coordinate. Note that this does not require that each pixel coordinate in an image must have a valid world coordinate. Similarly, not each valid world coordinate corresponds to a pixel coordinate.

To locate an astronomical source (celestial object) on the image, requires locating the pixel coordinate associated with the world coordinate for the source. This coordinate usually specifies the center of the source. However, the size of most sources is more than one pixel hence, a source might be associated with a set of pixel coordinates which locate it in the image data array.

3.2 Coordination transformations

Given a pixel position of an image, the computation of corresponding sky coordinates is illustrated conceptually in Fig. 2. Conversion from pixel coordinates to world coordinates is a multi-step process that includes linear and non-linear transformations. A summary of the transformations $\mathcal{T}_1$–$\mathcal{T}_5$ is shown in below:

- $\mathcal{T}_1$, Distortion correction
- $\mathcal{T}_2$, Linear transformation, translations, rotation, skewness, scale

[3] http://fits.gsfc.nasa.gov/, http://archive.stsci.edu/fits/fits_standard/

– $\mathcal{T}_3$, Distortion correction
– $\mathcal{T}_4$, Scale to physical coordinates
– $\mathcal{T}_5$, Coordinate computation per agreement

The mathematical details, including the interpretation of the intermediate world coordinates are described in [5] and [13]. An example of a non-linear transformation is a Distortion Correction. A Distortion Correction is a mapping (or mathematical formula) derived to reduce the effects of geometric effects such as the translation, rotation, tilt of the photographic plate, non-linearities in the coordinate measuring machine, physical effects such as refraction, aberration, precession and rotation, and optical defects such as radial and de-centering distortion. We do not expect distortion functions to have analytic inverses, because of the greater complexity of distortion functions [16]. We apply iterative methods to invert them. The result of such a process is probabilistic, and different precision levels can be reached.

3.3 Image transformations

The image transformational sequence (pipeline) of Astro-WISE is shown in Fig. 3. Each box in the figure represents an image transformational step. As an image moves through the transformational sequence, the pixel positions, pixels values, and image sizes will change due to rotations, alignment, scaling, distortions corrections and re-sampling effects. However, the world/physical coordinates will not change although the transformational effects might introduce new pixels coordinates due to the changes in image sizes. Mathematically, this introduces new world coordinates. Figure 4 illustrates the effect of applying linear shifts, rotations and distortion corrections during the image re-gridding process (Regrid Transform). The input grid (shown as small gray squares) is projected to the output grid (re-sampled image) represented by the large tilted/rotated ones. Based on a survey, the input to the final image may come from several images. For example, Several RegriddedFrames (from the Regrid Transform) created from ReducedScienceFrames (from the Reduce Transform) can be co-added (Coadd Transform) to form a deeper image with a substantial reduction of chip defects and divisions. In such cases the final

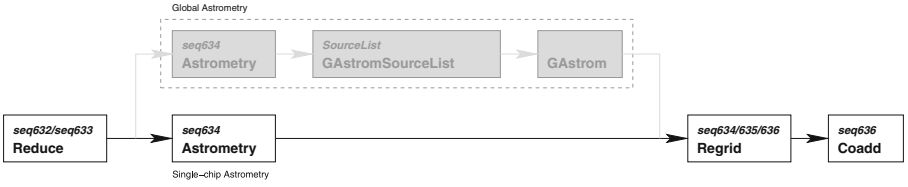


Fig. 3 The order and flow (*arrow directions*) of the image processing steps in the image processing part of a data reduction process

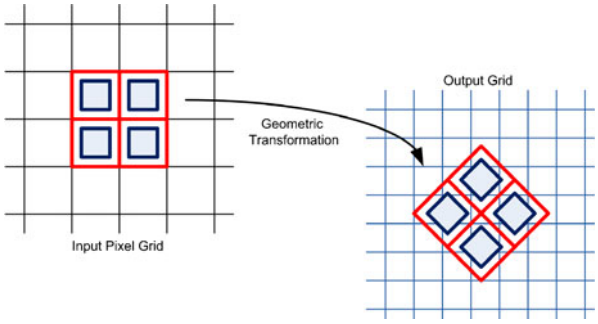


Fig. 4 An example of the mapping of an input image onto an output image (re-sampling). The figure was taken from [9] it describes the drizzling method. Pixels in the original input images are mapped into pixels in the sub-sampled output image, taking into account shifts and rotations between images and the optical distortion of the camera

value of a pixel in the CoaddedRegriddedFrame is a contribution from several other pixels from the various RegriddedFrames that were used to create the CoaddedRegriddedFrame.

Sources are extracted from the final transformed image by identifying connected pixels that exceed some threshold. When an interesting source is found through visualization, such as an outlier, an astronomer may want to know exactly which source this is and how its parameters were calculated. With data lineage such endeavours are supported. However the astronomer may want to know which pixels in the science and calibration images throughout the transformational process that contributed to the source. Finding the pixels that contributed to the source is pixel-lineage problem.

4 Pixel lineage framework

In this section, we formalize general data transformations and pixel lineage, then we briefly motivate why transformation properties can help us with pixel lineage tracing.

4.1 Transformations

We denote the set of all images as $\mathbb{V}$, one important observation that we leverage throughout the text is that every transformation/operation performed on an image (adding, subtracting, re-gridding and co-addition etc.) can be directly expressed as a (potential) function $\mathcal{F} : \mathbb{V} \to \mathbb{V}$. We define a transformation $\mathcal{F}$ as any procedure that takes an image I as input and produces image O as output. For any input dataset I, we say that the application of $\mathcal{F}$ to I resulting in an output set O, is denoted by $\mathcal{F}(I) = O$. Given transformations $\mathcal{F}_1$ and $\mathcal{F}_2$, their composition $\mathcal{F} = \mathcal{F}_1 \circ \mathcal{F}_2$ is the transformation that first applies $\mathcal{F}_1$ to I to

obtain I', then applies $\mathcal{F}2$ to I' to obtain O. Therefore given transformations $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_n$, we represent the composition as a transformation sequence $\mathcal{F}_1 \circ \cdots \circ \mathcal{F}_n$. For now, we will assume that all of our transformations are stable and deterministic. A transformation $\mathcal{F}$ is stable if it never produces spurious output items, i.e., $\mathcal{F}(\emptyset) = \emptyset$. A transformation is deterministic if it always produces the same output set given the same input set.

4.2 Pixel lineage

In the general case, an astronomical source lies on several pixels of an image. Although a world coordinate vector identifies each source, a source is associated with a small subset of pixels. Therefore a source o in the output set is derived from a small subset pixels from the input images.

Definition 1 Given a transformation instance $\mathcal{F}(I) = O$ and an output item $o \in O$, we call the actual set $I'' \subseteq I$ of input data items that contributed to o's derivation, the lineage of o. We denote it as $I'' = \mathcal{F}_l(o)$. The lineage of a set of output data items $O'' \subseteq O$ is the set $I'' = \{i \in I'' | \mathcal{F}(\{i\}) \in O''\}$.

Knowing something about the workings of a transformation is important for tracing data lineage. From lineage model presented in Section 2, we can explicitly assume that every output depends on every input and parameters passed to the modules. This is because of the black-box nature of the transformation programs that are used for computation. Therefore, such lineage accounts for an output product produced during the course of a dataflow execution by displaying a connected graph of data (image) dependencies (i.e., input, intermediate, and output data).

Based on the input and output images to a transformation, we can classify transformations based on how it maps input data items to output items. Using the relationships between pixel coordinates and sky coordinates, we then provide a tracing procedure for each classification.

4.3 Lineage tracing using known properties of astronomical image processing

In this section, we discuss how we use the relationship between pixel and world coordinates to trace lineage. We make use of known properties of astronomical image processing, where the pixel coordinates change as the image moves through a data-flow or pipeline while the world-coordinates do not change.

Consider a transformation $\mathcal{F}$ that inputs an image I and outputs an image O. Image I and O can be decomposed into a set of pixel and world coordinates $P = \{p_1, p_2 ... p_n\}$ and $W = \{w_1, w_2, ... w_n\}$ respectively.

Definition 2 Let P_I denote the set of pixel coordinates of an image I, and W_I denote the set of world coordinates of I. We define $\mathcal{T}$ to be a function that maps each pixel coordinate $p_i \in P_I$ value to its corresponding world coordinate $w_i \in W_I$ values such that $\mathcal{T}_I(P_I) \rightarrow W_I$ and $\mathcal{T}_O(P_O) \rightarrow W_O$.

Note that, in general, although $\mathcal{T}_I$ and $\mathcal{T}_O$ will be the same, the parameters used in the transformation will be different since the computation of pixel coordinates to sky coordinates is dependent on the specific frame and other related processing parameters e.g., projections. If the set of pixel coordinates associated to a source o on an image O is P'_O, then according to Definition 2, W'_O will be the world coordinates associated to o therefore $W'_O \subseteq W_O$. We are interested in finding W'_I, where $W'_I \subseteq W_I$ such that $W'_I = \{\omega | \omega \in W_I$ and $\omega \in W'_O\}$. The set W'_I is the set of world coordinates associated with the pixel coordinates in I that contributed to o's derivation. From Definition 1 the lineage of o is the set $P'_I = \{p_i \in P_I | \mathcal{T}(\{p_i\}) \in W'_O)$

In the remainder of this section, we consider three different transformation classes, based on how each transformation maps input pixels to output pixels.

4.3.1 CASE 1

A transformation $\mathcal{F}(I) = O$ is linear and one-to-one if each input data item produces exactly one output data item and if the pixel vector coordinate of an input data item $i \in I$ is the same pixel vector coordinate of the data item in the output image $o \in O$ where $\mathcal{F}(\{i\}) = o$. Let $\mathbf{p_j}$ be the pixel coordinate vector corresponding to a data item and $\mathcal{T}$ be a transformation on the space of coordinate transformations. If $\mathcal{T}_I(\mathbf{p_j}) = \mathcal{T}_O(\mathbf{p_j})\ \forall j$ and $\mathcal{F}(I) = O$, then transformation $\mathcal{F}$ is linear and one-to-one.

Representing an image I as an array and $\mathbf{p_j}$ is a pixel vector coordinate, then the notion $I[\mathbf{p_j}]$ will refer to the element in I at pixel position $\mathbf{p_j}$. If $\mathcal{F}(I) = O$, then the lineage of an element $o \in O$ at pixel coordinate $\mathbf{p_j}$ is the element $i \in I$ at pixel position $\mathbf{p_j}$. i.e., the lineage of $O[\mathbf{p_j}]$ is $I[\mathbf{p_j}]$.

4.3.2 CASE 2

The second categorization of a transformation is where each input data item from the input image produces one or more output data items (one-to-many) in the output image. Figure 5a shows an example of this transformation. For example a re-sampling function that maps each point in output image (O) to the average of that point and its neighbours in input image (I). The value of a point o in the output image is determined by the point i and all its neighbours in I depending on the kernel size. The lineage of an output item o according to this transformation is defined as $\mathcal{F}_l(o) = \{i \in I | o \in \mathcal{F}(\{i\})\}$.

We use O'' to denote the set of all pixels on which the object (source) lies. Using the terms already defined at the beginning of this section, if W'_O are the coordinates associated to O'' and let W'_I be the world coordinates associated to I'', If m is the number of points in W'_O, then for k=1,...,n, $w_k \in W'_I$ and for $k = (n+1), ..., m$, $w_k \notin W'_I$. Then $W'_I \subseteq W'_O$ according to this transformation. The set W'_I is pure but not complete. Not all points in W'_O are included in W'_I. Algorithm 1, is used to trace lineage of such a transformation.

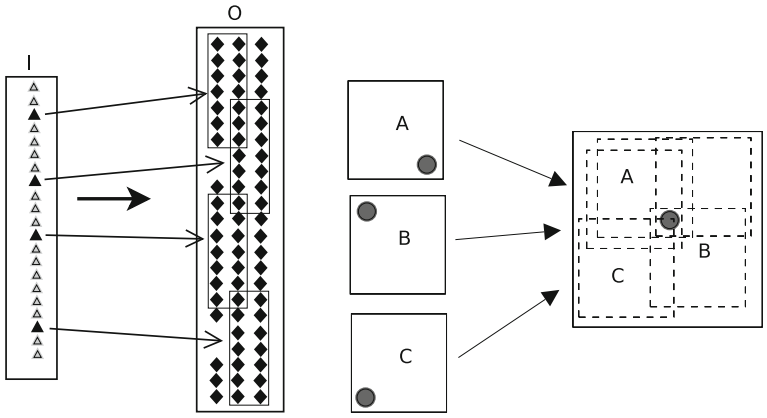
Algorithm 1 Tracing lineage for CASE 2 transformations

Input: T, O'', I,
Output: I''

$P_O \leftarrow$ pixel coordinates of O''
$W'_O \leftarrow \emptyset$
for $p \in P'_O$ **do**
  $w = \mathcal{T}_O(\{p\})$
  add w to W'_O
end for
$I'' \leftarrow \varnothing$
$P_I \leftarrow$ get all pixel coordinates for I
$W_I \leftarrow$ convert all pixel coordinates P_I to world coordinates
for $w \in W_I$ **do**
  if $\{w\} \cap W'_O \neq \varnothing$ **then**
    Add w to W'_I
  end if
end for
$I'' \leftarrow \mathcal{T}_I^{-1}(W'_I)$

4.3.3 CASE 3

The third type of transformation is illustrated in Fig. 5b. This is a transformation where one or more set of points in the input image are mapped onto the same point in the output image individually. The transformation



(a) Case 2 (b) Case 3

Fig. 5 Astronomical image transformations: an illustration of *CASE 2* transformation: each input data item from I, produces zero or more output data in O. An illustration of *CASE 3* transformation

`coadd` in Fig. 3 is an example of case three type of transformation. During the co-addition, several RegriddedFrames are co-added together to form a CoaddedRegriddedFrame. Note that transformation may also include one-one mapping at the sections where images are not overlapping. However, the pixel coordinates locating a data point in the input image is different from the pixel coordinate that locates the data point in the output image. Notice also from Fig. 5b that, the input data item (pixel) may come from one image or a combination of several images depending on the co-addition algorithm. We consider an example of finding lineage of all points that lie on the gray ellipse in output frame in Fig. 5b. These points could be located on one or more input images.

To trace lineage for this transformation, we begin by finding all images that were used to make the final regridded frame. After knowing the images, then we find those images that contributed to derivation of o. From these images we can find the specific pixels that contributed to derivation of o.

Let $\mathcal{F}(I) = O$, be a transformation over a set of images. We decompose I into unique disjoint partitions $\{I_1, I_2, ..., I_n\}$ such that $\mathcal{F}(I_k) = o_k$ where $K = 1, ..., m$. Then the set $O = \bigcup_{k \in K} o_k$.

If we take a set $O'' \in O$, we need to find which of the sets in $o_1, ..., o_m$ has elements in O''. This can be done by intersecting O'' with each $o_k \in O$. The sets where the intersection is not empty are the sets with points in O''. Then O'''s lineage is located on images I_k where $\mathcal{F}(I_k) = o_k$ and $o_k \cap O'' \neq \emptyset$. Then we can use the tracing procedure for case 2 type of transformations to find those elements (i.e., I''_k) that contributed to o'derivation. Let the set O'' be the set of points that contains a source o then $O'' \subseteq O$. Then the set I'' is the set all of input data items that contributed the results in O'', the Algorithm 2 is used to find I''.

4.4 Sky-to-pixel coordinate transformation

Note that, at the end of Algorithms 1 and 2, we require to find pixel coordinates for W'_I. Therefore to complete both algorithms we have to find the pixel coordinates for each of the world coordinate in the set W'_I.

The coordinate forward transformation is defined as $\mathcal{T} = \mathcal{T}_1 \circ \mathcal{T}_2 \circ \mathcal{T}_3 \circ \mathcal{T}_4 \circ \mathcal{T}_5$ (from Section 3.2). If $\mathcal{T}$ is invertible, then each $\mathcal{T}_i$ must have a well-defined inverse. For example, if $\mathcal{T}_i$ is the operation of applying a distortion correction while converting from pixel coordinates to corrected pixel coordinates, $\mathcal{T}_i^{-1}$ is the operation of removing the distortion correction. Because of the greater complexity of non-linear distortions corrections used during the conversion process and the non-linearities caused by distorted image coordinates, such distortion corrections do not have analytic inverses, therefore we relax the invertibility requirement and instead switch to heuristics. Since all the other transformations are linear, with the exception of $\mathcal{T}_1$ and $\mathcal{T}_3$, we omit the inversion steps for $\mathcal{T}_2$, $\mathcal{T}_4$ and $\mathcal{T}_6$ but we show the solution of finding the inverse of distortion corrections.

Algorithm 2 Tracing lineage for CASE 3 transformations

Input: $\mathcal{F}$, O'',
Output: I''

$P'_O \leftarrow$ pixel coordinates of O''
$\{I_1, I_2, ..., I_n\} \leftarrow$ I ;
$I' \leftarrow \emptyset$
$I'' \leftarrow \emptyset$
$W'_I \leftarrow \emptyset$
for each $I_i \in I$ **do**
  if $\mathcal{F}(I_i) \cap O'' \neq \emptyset$ **then**
    Add I_i to I'
  end if
end for
$W'_O \leftarrow \emptyset$
for $p \in P'_O$ **do**
  $w = \mathcal{T}_O(\{p\})$
  add w to W'_O
end for
for each $I_k \in I'$ **do**
  $P_I \leftarrow$ pixel coordinates for I_k
  $W_I \leftarrow \mathcal{T}_I(P_I)$ to world coordinates
  for $w \in W_I$ **do**
    if $\{w\} \cap W'_O \neq \varnothing$ **then**
      Add w to W'_I
    end if
  end for
end for
$I'' \leftarrow \mathcal{T}_I^{-1}(W'_I)$

For this work, we restrict our initial implementation to TAN Distorted tangential, which uses the plate solution, whose distortion correction polynomial is shown in (1). This is an example of one of the polynomials widely used for fitting measured offsets. The equation is a mapping between a source detected at pixel coordinates $\langle x, y\rangle$ in frame and its normal coordinates $\langle \zeta, \eta\rangle$ relative to the nominal centre of field. This degree of the polynomial p is determined by the nature of distortions being accounted. a_{ij}, b_{ij}, m and c are initially unknown coefficients, to be determined by the least squares analysis of the plate coordinates of the reference stars.

$$\zeta(x, y) = \sum_{i+j=1}^{i+j=p} a_{ij} x^i y^j m^k c^l ,$$
$$\eta(x, y) = \sum_{i+j=1}^{i+j=p} b_{ij} x^i y^j m^k c^l . \quad (1)$$

Equation 1 falls in non-linear kind of system of equations that can be expressed in the form of

$$\begin{aligned} \zeta(x, y) &= 0 \\ \eta(x, y) &= 0 \end{aligned} \tag{2}$$

In order to compute the $\langle x, y\rangle$ from $\langle \zeta, \eta\rangle$, we need to find the roots of the polynomial as expressed in (2). The functions ζ and η are two arbitrary functions, each of which has zero contour lines that divide the (x, y) plane into regions where their respective function is positive or negative. These zero contour boundaries are of interest to us. The solutions that we seek are those points (if any) that are common to the zero contours of ζ and η. We used Newton–Raphson method [15] to find the roots of the polynomial. This method gives us a very efficient means of converging to a root, if you have a sufficiently good initial guess. However, it can also spectacularly fail to converge, indicating (though not proving) that the putative root does not exist nearby.

4.5 Data lineage for a transformational sequence

Consider a simple sequence of two transformations, such as $\mathcal{F}_1 \circ \mathcal{F}_2$. For an input dataset I, let $I_2 = \mathcal{F}_1(I)$ and $O = \mathcal{F}_2(I_2)$. Given an output data item $o \in O$, if $I_2'' \subseteq I_2$ is the lineage of o according to F_2, and $I'' \subseteq I$ is the lineage of I_2'' according to $\mathcal{F}_1$, then I'' is the lineage of o according to $\mathcal{F}_1 \circ \mathcal{F}_2$. This lineage definition generalizes to arbitrarily long transformation sequences using the associatively of composition. Given a transformation sequence $\mathcal{F}_1 \circ \cdots \circ \mathcal{F}_n$, where each I_k is the intermediate result output from F_{k-1} and input to F_k, a correct but brute-force approach is to store all intermediate results $I_2, ..., I_n$ (in addition to initial input I) at loading time, then trace lineage backwards through each transformation at a time. This approach is inefficient due to the large number of tracing procedure calls when iterating through all transformations in the sequence. The approach also requires that input and output images (including intermediate results) are materialized at all times.

Fortunately, it is possible to relieve this problem by combining adjacent transformations in a sequence for the purpose of lineage tracing. The combination is based on the transformational properties. For example we do not have to iterate through consecutive transformations that are linear and one to one, and in some cases, if the input to the `coadd` transformation is one image, then the `regrid` and co-addition can be grouped into one transformation. This in general will reduce the overall tracing cost, while improving or retaining tracing accuracy.

4.6 Evaluation

We have implemented all of the algorithms described in this section in AstroWISE. Several tests have been carried out and are described in the following sections.

4.6.1 Results of pixel-lineage tracing procedures

We begin by testing the accuracy of the sky-to-pixel inverse coordinate transformations algorithm. (i.e., $\mathcal{T}^{-1}$). We selected 90 RegriddedFrames and we created a SourceList (source catalogue) for each frame. Attached to each source in a SourceList, is its sky coordinate (α, δ) and its pixel $\langle x, y\rangle$ coordinate. The pixel coordinate specifies the location of the source on the image where the source was detected. During the creation of the SourceLists, the sky coordinate values are computed from pixel coordinates via a forward coordinate transformation process. In this test, for each (α, δ), we try to compute the corresponding (x, y) through an inverse coordination transformational

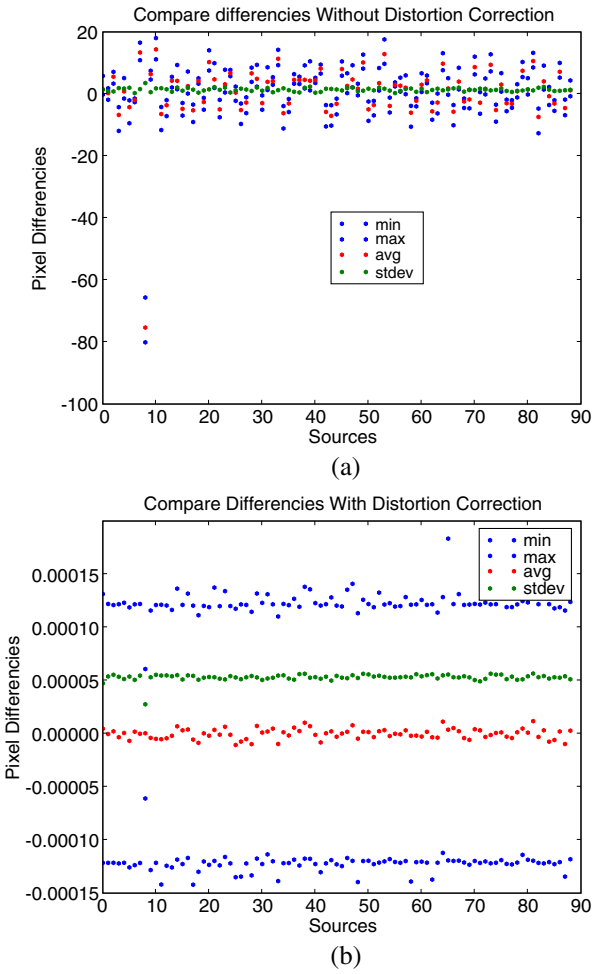


Fig. 6 The results of the coordinate transformation test. The difference in pixel coordinates (x,y) without and with (respectively) the distortions accounted for in the inverse transform (world to pixel). *Blue* is min/max difference for each SourceList, *red* the average and *green* the standard deviation. Panel **b** shows a positional accuracy improvement of about five orders of magnitude over panel **a**

process. The values obtained from this computation are compared against the values stored for each source in the SourceList.

Two tests were done to test the inverse coordinate transformational algorithm. The first test did not include the distortion correction, while the second test included distortion correction. The result of this test is shown in Fig. 6a and b. The figures show the difference in pixel coordinates without and with (respectively) the distortions accounted for in the inverse transform (world to pixel). Blue is min/max difference for each source list, red the average and green the standard deviation. Figure 6b shows a positional accuracy improvement of about five orders of magnitude over Fig. 6a. This is an indication that we can accurately find corresponding $\langle x, y \rangle$ positions of pixels in an image. The algorithm can be tuned to achieve any precision, but it is a trade-off between performance and accuracy. A higher precision will increase on the time the algorithm takes to converge (to find the root).

The two images in Fig. 7a and b also show what would happen if the reverse transformation does not include distortion corrections in locating the sources. The beginning point is a source with a set of sky-coordinates, and the task is to locate the corresponding source on the raw science frames. Notice in Fig. 7a that there is an offset in the positional accuracy of the three sources marked with ellipses, which is corrected in Fig. 7b after implementing the correct reverse transformation which includes distortions.

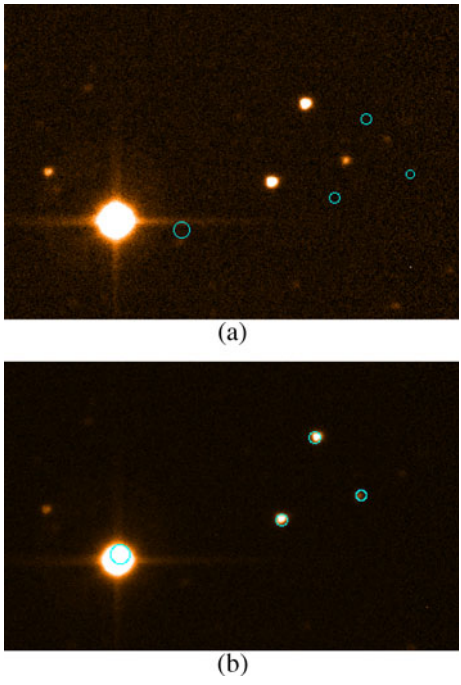


Fig. 7 The figure in panel **a**, shows the results of mapping sources into an image without considering the distortion correction, whereas panel **b** considers the distortion correction while mapping sources back to the raw images

4.6.2 Lineage retrieval

Astro-WISE implements the dependency cutout service which is available on [6]. The service extracts pixels of a source, as rectangular smaller images from all images that were used during the processing of the source. The service follows lineage in the system to find all dependent images and uses pixel lineage to find the actual location of the source on each image. Figure 8 shows an example of image cutouts of some of the images that were created or used during the processing of one of the sources.

4.6.3 Data rates

In Astro-WISE, we recognize that data lineage is also first-class data and that data lineage can be used to simplify and partially automate the scientific data analysis cycle. Lineage forms part of the system and as such, the target processor (data processing engine of Astro-WISE) follows dependencies in the system to assemble pipelines for processing data. Computations in Astro-WISE are data intensive and do generate many intermediate datasets. Dependencies exist among intermediate datasets and these are recorded and stored. Dependencies depict derivation relationships between intermediate datasets. The existence of intermediate data simplifies data processing and speeds up execution times, since this eliminates processing steps for intermediate data products that have already been generated by another process.

For everything processed in Astro-WISE, we store all images and processing data generated during the data flow to centralized database. Using this information we can quickly compare how many times an object has been reprocessed or how many objects refer to it. For example, Fig. 9 shows the number of versions of Bias Frames that were derived from a fixed set of RawFrames as a function of time. The output shows that different RawFrames (per CCD) have been reprocessed a number of times even though they belong to the same image. The y-offset of 6 at the beginning means that six raw images are considered in this example.

For purposes of this work, we are interested in the lineage of a source. Let us consider the RawFrame and a list of sources as the only data products that

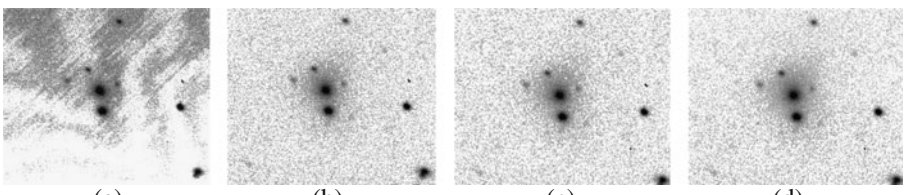


Fig. 8 Image cutouts of one of the sources processed in awe. The image in panel (**a**) is the RAW, (**b**) is the RED, (**c**) is the RGR, (**d**) is the COADD. These were used/created in the image pipeline while processing the source

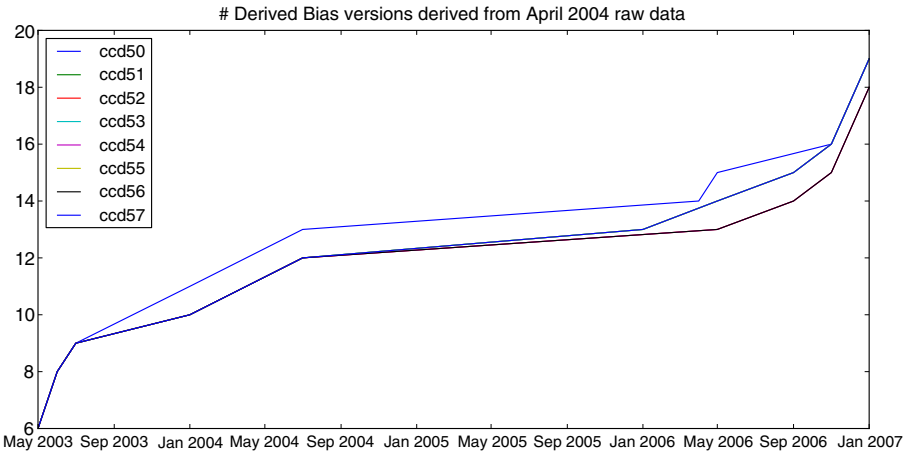


Fig. 9 Graph showing number of BiasFrames (on the *y*-axis) that were derived from a fixed set of RawBiasFrames as a function of time (*x*-axis). This example shows different CCD's (shown in the legend) that have been reprocessed different number of times even though they belong to the same image

are not part of lineage data, then any other data product (e.g., metadata, image files, other data objects) will be lineage data. From [21], data lineage is stored as persistent objects in a database. Lineage data is stored as fully integrated objects or as descriptors. Only pixel values are stored outside the database in image and other data files. These data files are registered in the Astro-WISE metadata database with the unique filename. From this model, the database has a record of the processing history for a source. Each object stored in the database is identified uniquely with an `object_id` (OID). Counting the OIDs will give us an idea of how much data is stored as lineage data. The size of a science image is dependent on the instrument and perhaps the survey. Since size is variable, we cannot give a generalization of data files sizes.

Catalogues created from images are stored in Astro-WISE as a SourceList object. These catalogues consist of sets of sources and parameters (or attributes) that quantify properties of these sources. Normally, the catalogues are derived from a processed frame existing in the system, but this is not a requirement.

A dataflow usually has a specification that serves as a template for executions together with set of valid runs for the given specification. Informally, a dataflow specification consists of a set of different modules and defines the order in which they can be executed. In Astro-WISE, a SourceList can be created from a ReducedScienceFrame, RegriddedFrame or a CoaddedRegriddedFrame. Creating a SourceList from a ReducedScienceFrame, although considered a valid run, will skip the steps of creating the RegriddedFrames and CoaddedRegriddedFrame. For the examples in Table 1, we selected SourceLists created from CoaddedRegriddedFrames. Since these represents

Table 1 Data lineage objects for each SourceList object

SourceList ID	RawFrames used	OIDs
10011	40	5,669
115	48	4,742
116	40	3,702
117	40	3,168
12817	72	7,992
12837	96	10,455
12838	72	8,215
12857	72	9,124
12858	72	8,182
12859	72	6,411
12877	32	7,917
12878	64	7,690

a complete run where all modules in the specification were executed. A CoaddedRegriddedFrame can be made from several RegriddedFrames (second column of the table).

For each SourceList, we count how many objects exist that are stored as data lineage (i.e., the column labeled OIDs). The OIDs give us an indication of how many objects are stored as lineage data for each source. This is dependent on the number of frames used to create the SourceList. However if two SourceLists refer to the same dependencies, then the SourceLists will share some of the OIDs.

5 Sub-image processing

Practically, sub-image and full-image processing follow the same standards, data quality and processing requirements. However processing parameters, attributes and some modules might defer. This is because pipelines have been written and designed for instruments with fixed detector properties (e.g., image size, calibration frames, overscan regions, etc.). All metadata and processing parameters are based on an instrument or a detector. These variables must be modified to process a sub-image.

Since no metadata (processing data) exists for sub-image processing, we use analogical reasoning to infer the necessary changes and then apply them to processing. By matching and retrieving existing pre-processed information (data lineage) in the system, we know what relationship exists between what we want to process and what has been processed before. Using the relationships, we are able to determine the difference between pipelines (and objects). Hence, we are able to assemble new pipelines so that the new processing follows the new user processing requirements for a particular region on the frame.

During data analysis work cycles, users often have to integrate new features into existing pipelines. For example, a user may wish to improve a given result by adjusting parameter or changing a method/algorithm. In either case, there usually exists an example based on full-image processing that demonstrates

the given technique. The underlying assumption is that the source has been processed before as part of a full image and a user would like to carry out a detailed analysis or a computation to a source that lies on few pixels of an image. Algorithm 3 summaries sub-image processing.

Algorithm 3 High-level view of the algorithm used for Sub-image processing

Input: Object to be re-processed
Output: Re-processed object
if Object exists in database **then**
  Select pipeline that was used to process the object (Section 5.1)
  for every node in the pipeline **do**
    Query database for dependencies
    Check if dependencies are `uptodate()` and if they `exist()` (Section 5.2)
  end for
  Edit pipeline for sub-image processing (Section 5.3)
  Make new parameter selections/changes (Section 5.3.1)
  if dependency is an image **then**
    Compute BoundingBox (Section 5.4)
    Check CoverageMap if region has been processed (Section 6.1.1)
    Make dependency cutout (Section 5.4)
  end if
  Re-analyse dependencies
  Process sub-image(s)
  Update CoverageMap (Section 6.3)
  Run `SexExtractor` on the processed sub-image to extract source (object)
end if

5.1 Selecting pipelines

The basic idea behind our algorithm is to search for a graph representation of the dataflow that was used in earlier runs to process the full-image from which the sub-image to process will be extracted. The next step is to retrieve data lineage and intermediate data products produced by this graph from the database. Then the retrieved data will be used as input to the sub-image processing pipeline. However, since we are processing a sub-image, selected lineage data to be used as input to the sub-image pipeline will have to be modified. When such changes occur then the part of pipeline affected by the changes must be re-run. The re-run will consider dependencies and only execute those parts of the pipeline affected by the changes.

During a typical data analysis, in the process of trying different datasets, trying different techniques, and tweaking parameters, a user may create

multiple sources. Creating multiple sources also implies there are several different pipelines that were used to create the sources. We refer to these pipelines as candidate pipelines. It is clearly impractical to locate a pipeline that matches a certain criteria by examining each candidate pipeline individually.

To select a pipeline in Astro-WISE, a user selects particular parameters that contain the desired attributes. These parameters are used to build a query. A query interface which is used for processing an object, can also be used for building a query. Once a query represented as a pipeline fragment is constructed, we can use the pipeline matching algorithm on each candidate pipeline to determine if it satisfies the query. A candidate pipeline that has more matched elements as specified in the query pipeline will be selected.

5.1.1 Matching pipelines

Pipelines arrange the processing of objects by means of a graph that defines the dependencies and dataflow between objects. In our pipeline representation, a directed edge will point from object X to object Y, if object X depends object Y. With a graph for each pipeline, we compute the mapping by comparing objects (nodes). For each node in the graph, we create a *set* of its dependencies. The elements of the *set* are all edges pointing from the node. Let G_p and G_q be graphs corresponding a candidate pipeline and the query fragment. If N_p is a node in G_p and N_q is a node in G_q, we define *score* S that measures the pair wise similarity of each node as, the number of matched elements in N_p and N_p, divided by total number of distinct elements in N_p and N_q.

Algorithm 4 Selecting a candidate pipeline

Input: list of sources, pipeline fragment/query pipeline
Output: candidate pipeline $G_c(N_c, E_c)$

```
S ← list of sources
G_q ← query pipeline with selection parameters
G_c ← candidate pipeline
S_max ← 0
if length of S ≠ 0 then
  for each source s_i ∈ S do
    G_p ← query Astro-WISE for pipeline that was used to process s_i
    S_i ← match G_p with the query pipeline G_q
    if S_i > S_max then
      S_max ← S_i
      G_c ← G_p
    end if
  end for
end if
```

A *score* of zero implies, there are no matching elements in the pair wise nodes. It could also mean that the node in the candidate pipeline was not specified in the query pipeline. However, we restrict the comparison to only those nodes specified in the query pipeline. The higher the score, the more likelihood that there are more matching elements between the nodes. The overall score of a graph is the sum of all scores on all nodes. The graph with the highest score will be selected. Selection of the candidate pipeline is summarised Algorithm 4.

5.2 Pipeline building

Astro-WISE uses the advantages of Object-Oriented Programming (OOP) to process data in the simplest and most powerful ways. In essence, it turns the aforementioned data objects into OOP objects, called process targets (or Process- Targets). These are instances of classes with attributes and methods that can be inherited. For the remainder of this document, the class names of objects, their properties, and methods will be in `teletype` font for more clear identification.

The starting point of sub-image processing is the selection of the target, i.e., set of sky positions and a set of parameters. The system matches and selects the pipeline from all candidate pipelines as described in Section 5.1. Each node in the pipeline is associated to an object. Each object is identified with a unique object ID (OID). The OID is used as a key when searching the database for data lineage information related to a particular object. Each OID is also associated with a specific instantiation of a class (module). Using this unique ID, we can query for all data that went into the processing of this object. If the OID is associated with an image, a cutout is made of the pixels of interest from this image and used as input to the module. The pixels extracted as a cutout are determined through pixel lineage.

Astro-WISE then analyses this graph to detect modules that must be rerun, based on the processing changes required for sub-image processing. If the requested target already `exists()`, (i.e., if it has been processed using parameters and input data as required for sub-image processing) the system will check all its dependencies up to the raw data. If all its dependencies are `uptodate()` then the target object is returned. If all or some dependencies are not `uptodate()`, then these dependencies will be made on-the-fly, by calling the dependency's `make()` method. If new versions exist of the classes that created the dependencies, then these dependencies will be re-made using the new-versions of the classes. This analysis is implemented with three key methods:

– Each target has a `make()`[4] method which explicitly specifies file targets, their dependencies, and modules/methods required to process the target.

[4]The `make()` method follows the `Linux` make metaphor.

- The `exist()` method queries the database, using a set of parameters. The set of parameters depends on the class. If the query results in multiple objects, the most recent is returned.
- The `uptodate()` method, first checks if the object is flagged, if so the object is out-of-date. Then all the dependencies of the object are checked if they are the newest versions. Essentially calling the `exist()` method on every dependency. If not all dependencies represent the newest version, the object is out-of-date. The last check is to call the `uptodate()` method on all the dependencies. This recursive calling of the `uptodate` method can be done to a certain depth, or until the last dependency in the processing chain.

5.3 Pipeline changes

The matched and retrieved pipeline has to be modified to support sub-image processing. This also includes modifying parameters and input data (e.g., use image cutouts rather than full images). We define a function Δ as a function that takes in input, the matched pipeline G_i and transforms it to another pipeline G_s (pipeline for sub-image processing). Assuming $\Delta(G_i) = G_s$, it is clear that Δ is not unique for each processing. Our ultimate goal is to find Δ. i.e., the changes required to transform G_i to G_s.

For each module or pipeline, we identify all tasks that are required to be changed or modified to support this new framework. Those tasks are then included in the system, as a new module, attribute or parameter without changing the overall data model. If we denote the set of all pipelines as $\mathcal{P}$, every operation performed on a pipeline, (e.g., adding parameters, modules, etc.) can be directly expressed as a (potentially partial) function $F : G \rightarrow G$. Then Δ is defined by such functions. Our results depend on making these functions part of Astro-WISE. These functions are dynamically loaded whenever a user requests to process a sub-image. The changes made to the modules for image analysis are specific and are beyond the scope of this chapter. We present two examples to demonstrate the kind of changes required for sub-image processing:

- **Astrometric Parameters**: A critical step for astronomical processing is deriving an 'Astrometric solution'. This is derived by fitting distortion polynomials to images, taking into account the objects seen. For accurate results, several reference stars are used to derive the final solution. For the case of sub-image processing, such a process would fail since reference stars on the sub-image will be very few. In Astro-WISE, Astrometric parameters are each linked to a single processed science observation (a single detector chip of one exposure). It is that observation that provides the source positions to be calibrated via the astrometric solution. Using data lineage we locate the astrometric solution that applies to the region we are processing. The solution is then modified and fitted to the pixels of the sub-image.

– **Overscan Correction**; The overscan is a number of rows and columns created by doing a few empty readout cycles before the detector is read. They appear as blank regions attached to the image and serve as an individual BIAS that comes along with every image. A BIAS shows the electronic noise of the camera and possible systematics. It represents the actual state of the camera at the time the exposure was taken. Each image has overscan rows and columns attached to it. During data reduction, each image must be corrected for an overscan. The overscan correction is done by smoothing the overscan regions and subtracting it from the regions with data. After this correction, the overscan region is trimmed off. For the case of sub-image processing when extracting the sub-image from the full-image the algorithm must be aware of the overscan regions. The overscan regions corresponding to the sub-image are also extracted and used to perform an overscan correction on the sub-imageThe trimming operation is not done in this case since the sub-images do not have overscan rows and columns.

5.3.1 Parameter and attribute selection

The example below displays a lineage tree for any object. This particular example shows the parameters that were passed into a program called Swarp (Use for Co-addition)

```
awe> regrid = RegriddedFrame()
awe> regrid.swarpconf.info()
SwarpConfig: <astro.main.Config.SwarpConfig object
              at 0x1d893ad0>
                 +-CELESTIAL_TYPE: NATIVE
                 +-CENTER: ([111.428571428571,
-0.95744680851063801]
                 +-INTERPOLATE: N
                 +-OVERSAMPLING: 0
                 +-PIXEL_SCALE: 0.2
                 +-PROJECTION_TYPE: TAN
                 +-RESAMPLING_TYPE: LANCZOS3
                 +-SUBTRACT_BACK: N
                 +-object_id:
'83A31678D4D49F0AE0407D81E60E38BA'

                ---- etc -----
awe>
```

To change any parameter during processing, the command below would suffice,

```
awe> regrid = RegriddedFrame()
awe> regrid.swarpconf.RESAMPLING_TYPE = 'NEAREST'
```

In addition, we have also defined a web interface exemplified in Fig. 10 which shows the parameters that were used during processing. The same interface can be used change parameters as required for sub-image processing.

5.4 Dependency cutouts

This service extracts or "cuts out" rectangular regions of contributing pixels from some larger image returning a sub-image of the requested size. Contributing pixels are the pixels that contributed to the derivation of a source. Extracting image cutouts is according to the VO standards described in [27]. Cutouts in Astro-WISE are drawn from a collection of images that were used or produced during a specific dataflow.

A science image has an associated weight frame and/or calibration frames. Weight frames are used to assign individual weights to every pixel in the science image depending on the reliability of the information carried in the pixel. In the same way, calibration frames are used to correct pixels in the science image. There is usually a mapping between the pixels in the weight frame and/or calibration frames to the corresponding science frame. A lineage query for a source will lead the information system to the science and calibration images that were used to process the source. Pixel-lineage will lead the information system to the specific contributing pixels in the images. Cutouts are made for only the contributing pixels.

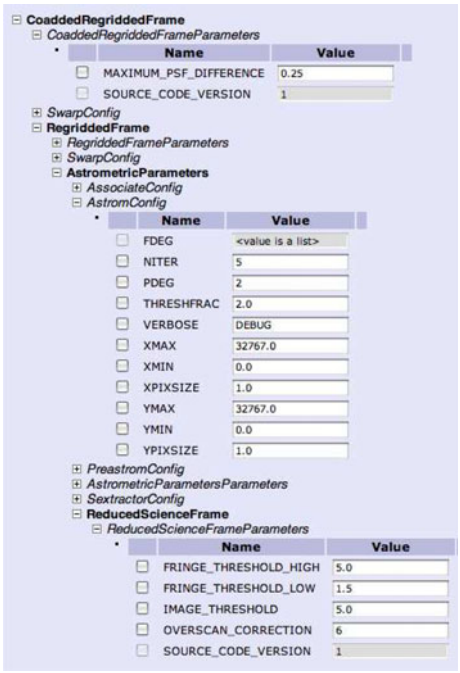


Fig. 10 A web interface for parameters selection

The size of the cutouts is defined by BoundingBox(BBox). The BBox is a tuple of pixel coordinates $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$, where $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$ are lower and upper pixel coordinates of a rectangular region respectively. To determine the size of the BBox, we begin with the position of the source, then we follow the procedure outlined below:

(i) find source with *Ra* and *Dec*
(ii) get its size on the sky;
(iii) scale it by the pixel scale factor
(iv) convert into pixel coordinates;
(v) draw a rectangular section that encloses all pixel coordinates computed in step (iv)
(vi) get minimum $(x_{\min}, y_{\min})$ and maximum $(x_{\max}, y_{\max})$ values of the pixel coordinates in the rectangular section
(vii) $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ is the BBox.

Using the $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$ together with the pixel lineage tracing procedures, we work through all image dependencies while computing the BBox on each frame. However, Astro-WISE implementation allows a user to specify the size of the BBox or the size of the cutout. This may be specified using one or two values in the coordinate space. These values are used to extract the cutout with reference to the position of the source.

6 Book keeping

We describe the management of image cutouts and how Astro-WISE keeps track of dependencies during sub-image processing. We introduce key classes that enable the methods presented in this section:

- `DBObject`: This is the base class of all persistent objects. It defines persistent properties that all classes share. For example, the object identifier (OID) which uniquely identifies each object in the database.
- `DataObject`: This class is derived from `DBObject` and is the base class for persistent classes of which their instances have an associated file on the data-server. It defines the persistent property `filename`. The `filename` attribute specifies the name of the file as it is stored on the data-server. Each file in Astro-WISE has a unique name and this name is stored in the database as part of the metadata. Every instance of the class `DataObject` or every class which is derived from class `DataObject` has an associated data file.
- `FileObject`: Data files are stored in Astro-WISE as `FileObjects`. An instantiation of the `FileObject` class creates the StorageTable. The OID and the object type of a `DataObject` are used as reference to identify the relationship between the data file and the `DataObject`. Figure 11 shows a UML class diagram that shows the relationship between the `FileObject` and `DataObject`. A `FileObject` contains information of a file, such as

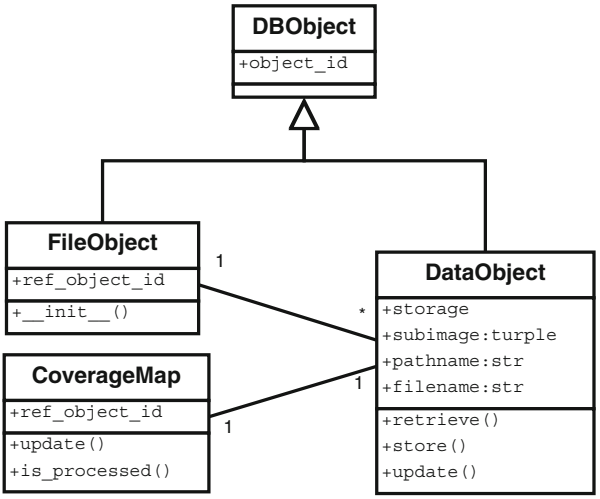


Fig. 11 Class diagram to support storage of multiple processed sub-imagess to the same `FileObject`

file size, creation date, hash value, etc. An abstract `Storage` class defines a general interface for the different protocols. It defines a `retrieve` and a `store` methods which should be implemented by the different kinds of storage, that is, classes that inherit the `Storage` class.

– `BaseFrame`: This is the base class for all classes representing image data. A `BaseFrame` is a persistent FITS file object. `BaseFrame`, inherits from `DataObject`. All objects derived from `BaseFrame` describe the contents of associated FITS images and have methods that operate on these images. It is from the `BaseFrame` class from which more complicated image classes such as the RawScienceFrame, ReducedScienceFrame, are derived.

6.1 Smart processing

The classical paradigm to handle data streams of large physical experiments is characterized as data-driven and feed-forward [28]. Operators have a task to push input data through the stream, often by means of a pipeline irrespective of whether the derived data items are actually used by the end users or have been derived before. In Astro-WISE, we use a different approach. Firstly, we limit the creation of data objects to those required for an experiment, secondly we support re-using existing data objects rather than recreating them as long as input parameters and other dependencies have not changed.

With sub-image processing, the same applies. Users can now focus on processing regions that have sources of their specific interests, rather than processing the full frame and discarding what does not interest them. Similarly, a region on a frame should not be re-processed if it has been processed before. Especially if the processing parameters and dependencies have not changed.

To solve the problem of finding which region has been processed, we introduce the idea of a CoverageMap as presented below:

6.1.1 CoverageMap *(*CMap*)*

A CMap is a way of representing which regions on an image have been processed. The CMap is implemented as a pixel-map in the system and is derived from the `BaseFrame`. There is a one-to-one mapping between the pixels of the CMap to the pixels of the `BaseFrame`. Each pixel is assigned a value of 1 for good/processed or 0 for bad/unprocessed pixel in the CMap. Figure 12 represents an example of a CMap, with regions R_1 to R_9 representing processed regions in a frame. If a user requests to process a region on a frame, the CMap is checked to establish if that region has been processed fully or partially as follows:

Given a binary $2\mathcal{D}$ image (CMap in this case) with regions R in $\{R_1, R_2, ...R_k\}$, where R_i represents all processed regions in R. If R_i is a set of points (x, y) over R, given an arbitrary region R', we find if any points (x, y) in R' that do (not) exist in any R_i. This is implemented as shown in Algorithm 5. If the region has been processed before then a link to the processed frame is returned, else a False is returned implying that the region or part of the region has not been processed before.

6.2 Cutout dependencies

Any module in Astro-WISE that uses an image as one of its inputs, does processing on the image and outputs another image, will store any new images created and create links to the images used and to all other dependencies.

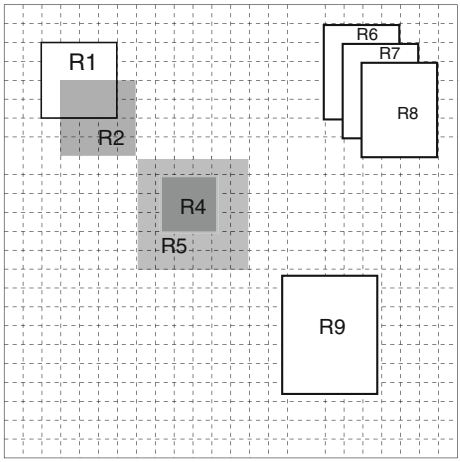


Fig. 12 An example of a CMap. Regions *R1–R9* represent processed regions. These regions will have a value of '1' in the pixel-map

Algorithm 5 Checking for processed regions on a frame

Input: CMap, `BaseFrame`, R'
Output: Boolean, True or False, if True, link to processed frame

```
if BaseFrame has a CMap then
  R ← R_1, R_2, ....., R_k,
  for R' ∈ R do
    if R' ⊂ R_i then
      Return Link
    else if R' ⊃ R_i then
      Return False
    else if R' = R_i then
      Return False
    else if R_i ⊃ R' then
      Return Link
    else if R' ∩ R_i then
      Return False
    else if R' ⊃ R_i then
      Return False
    else
      continue
    end if
  end for
end if
```

In case the input images have not been stored before, `DataObjects` for the images will be created, stored and linked to the results.

In the case of sub-image processing, for every sub-image we process, a new `DataObject` is created. A store command would want to store the cutouts used as input to the processing module and also store the newly processed sub-image. However, since cutouts are made from existing images, the storage of the cutouts will duplicate pixel data in the system and will make history tracking complicated.

To avoid duplication of pixel data in Astro-WISE, image cutouts that used as input to any transformation are not stored. We instead store a link that refers to the image from which the cutout was made and make the cutout parameters persistent. This implies that all sub-images processed from the same frame will refer to the same dependencies. To avoid the storage of the cutouts, we defined three attributes and redefined the `retrieve` method:

1. `subimage` attribute specifies the BBox.
2. `filename` attribute refers the name of image file
3. `pathname` attribute specifies name and path of the filename as stored on the local processing node. This attribute separates into a filename which is persistent and a file-path, which is not persistent. This is because the pathname is only meaningful on the site where a process is running.

4. `Retrieve()` is used to transfer data files from the data-server to the local node for processing. For sub-image processing, the `retrieve` method implements the cutout algorithm in addition to file renaming. `Retrieve` uses BBox parameters which specified by the `subimage` attribute to implement the image cutouts algorithm and the retrieval of cutouts.
   For example, if a user wants to retrieve a file associated with a `DataObject`, he can use the `retrieve` method from the `DataObject` class to retrieve the file. The `DataObject`'s `retrieve` method in turn calls the corresponding retrieve method from the `Storage` class. In the context of the sub-image processing, the retrieve method returns a cutout of the image file associated with the `DataObject`. In addition, the retrieve method renames the retrieved cutout and the new name is referred to with the `pathname` attribute.

If a `DataObject` has an attribute `subimage`, then the `pathname` refers to the name of the sub-image and `filename` attribute refers to the BaseFrame from which the cutout was made. Otherwise both the `filename` and `pathname` will refer to the `BaseFrame`. Since the filename still refers to an existing object, with an existing OID, such an object will never be (re)stored.

6.3 Storage of processed sub-images

Assuming that a user wants to process several objects which lie on the same BaseFrame, the user might decide to make cutouts and process each individual object separately. If such a process leads to the creation of a ReducedScienceFrame, then a several ReducedScienceFrame objects, i.e., several `FileObjects` and `DataObjects`, all linked to the same RawScienceFrame will be created. Even if they share the same input datasets, parameters and methods. This is not only wasteful but will complicate data management and history tracking.

A processed sub-image may be stored as an individual processed image, or stored in some larger image. If several sub-images are processed from the same BaseFrame, then all pixel processed data will be stored in the same `FileObject`. A `FileObject` is created for each processing parameter set. A processing parameter set contains similar parameters that were used during processing. The pixels of the processed sub-images will be stored in the same `FileObject` provided they all share the same processing parameter set. In case of a parameter change, a new `FileObject` will be created to store pixel data for the processed sub-image. A conceptual example of a `FileObject` used to store processed pixel data is shown in Fig. 12. Assuming regions R1 to R9 represent processed regions, then only those regions will contain valid data. We then use a CMap to assign individual weights for every other pixel which does not lie in any of the processed regions.

7 Evaluation

7.1 Detailed use-case

The use of sub-image data processing proves to be useful for a number of scientific use cases. We show one of the possible implementations of the developed sub-image processing.

In the search for special types of objects (quasars, brown dwarfs, white dwarfs etc.), the scientist usually works with multi-band catalogues created from a number of images. Objects which are selected as candidates for a class of objects the scientists is interested in are called "drop-outs". A drop-out is an object which is missing one or more detection values on the images used to create a catalogue. For example, if the same detection threshold is used for all images in different filters used to create a final multi-band catalogue, the brown dwarf will appear on infra-red images but will be most probably missed on ultra-violent and visual bands.

For this work we used *W3* and *W4* of CFHTLS [24] ingested in Astro-WISE. CFHTLS is a deep sky survey observed by Canada-France-Hawaii Telescope in *u,g,r,i,z* bands.[5] *W3* and *W4* fields of *CFHT* Legacy Survey was combined with the corresponding area in the VISTA Kilo-Degree Infrared Galaxy Survey (VIKING[6] survey). VIKING is an ESO public survey in *Y,Z,J,H,K* bands. The reduced images were provided by Cambridge Astronomy Survey Unit[7] and catalogues were extracted in Astro-WISE.

This is seven square degree area which is covered in ten filters (*u,g,r,i,z*—for CFHT, *Z,Y,J,H,K*—for VIKING). The resulting catalogue contains 2,247,126 sources, 1,801,216 of them are formally dropouts. However only a small fraction of these objects can be classified as candidates for some type of special objects. For example, approximately 1,000 of the sources can be brown dwarfs, depending on the selection criteria. The important task for all these sources is to estimate their minimum flux. To do so without the sub-image processing, we have to repeat the source extraction for a full image with parameters defined from the images where the source was detected. With sub-image processing, we can carry out source extraction for a particular source.

`SExtractor`[8] which is integrated in Astro-WISE is the program that extracts sources and computes source attributes. The program estimates the magnitude of the object using shapes defined from images where the object was detected. For this task we use `SExtractor` in the so-called double-image mode. This requires two images, the first image (detection image) will be

[5] http://www.cfht.hawaii.edu/Science/CFHTLS/

[6] http://www.eso.org/sci/observing/policies/PublicSurveys/sciencePublicSurveys.html

[7] http://casu.ast.cam.ac.uk/

[8] SExtractor is a program that builds a catalogue of objects from an astronomical image. http://www.astromatic.net/software/sextractor

Table 2 The table demonstrates the missing magnitude values of particular filters

SLID	SID	RA	Dec	*u*-mag	*g*-mag	*r*-mag	*i*-mag	*z*-mag	*Z*-mag	*Y*-mag	*J*-mag	*H*-mag	*K*-mag
1686281	1,643,576	36.2	−5.27	21.05	21.06	20.74	21.68				25.11	24.66	
1686281	1,640,449	36.47	−5.37	20.26	20.6			23.98	24.74	24.61			
1686281	1,455,855	35.63	−4.19	20.7	20.86		22.13		25.16				
1686281	79,615	34.13	−5.44	18.7	18.84			23.19	23.91	23.17	24.09	25.13	

We later use the filters that returned magnitude values to estimate magnitudes values for filters that did not return any values

Table 3 Re-computed magnitude values for selected sources

SLID	SID	RA	Dec	*u*-mag	*g*-mag	*r*-mag	*i*-mag	*z*-mag	*Z*-mag	*Y*-mag	*J*-mag	*H*-mag	*K*-mag
1686281	1,643,576	36.2	−5.27										24.18
1686281	1,643,576	36.2	−5.27							24.52			
1686281	1,643,576	36.2	−5.27						25.55				
1686281	1,643,576	36.2	−5.27					[a]					
1686281	1,640,449	36.47	−5.37										24.51
1686281	1,640,449	36.47	−5.37									23.74	
1686281	1,640,449	36.47	−5.37								25.70		
1686281	1,640,449	36.47	−5.37				26.25						
1686281	1,640,449	36.47	−5.37			[b]							
1686281	1,455,855	35.63	−4.19										22.29
1686281	1,455,855	35.63	−4.19									23.71	
1686281	1,455,855	35.63	−4.19								99		
1686281	1,455,855	35.63	−4.19							23.73			
1686281	1,455,855	35.63	−4.19						26.14				
1686281	1,455,855	35.63	−4.19			99[c]							
1686281	79,615	34.13	−5.44										22.10
1686281	79,615	34.13	−5.44				23.90						
1686281	79,615	34.13	−5.44			24.60							

A row in each table of the same source indicates a unique process that was used for source extraction
[a]There is no overlap between ReducedScienceFrames for *i*-mag and *z*-mag
[b]Error in Projections
[c]No source detected at this position

used for detection of sources, and the second image (measurement image) for measurements only. The detection image will define the shape and other parameters which shall be used to estimate the magnitude of the source in the measurement frame.

Rather than using the full-image for this, we make cutouts of the sources in the detection and measurement frames. Other parameters, which are selected from lineage data, have to be modified to reflect the changes in image sizes that are being passed into the pipeline. It is very important that cutouts are precise and accurate to avoid using parameters that could be detected from wrong pixels. This procedure is summarized below.

- We begin by searching the catalogues (or `SourceList` the table identifier in Astro-WISE) for all sources that fall in the description of dropouts stated above. For purposes of this demonstration, only foursources have been selected. These are shown in Table 2. The sources with SLID = 1686281 and SID = 1643576 from the table has missing magnitude values in the *z-mag, Z-mag,Y-man* and *K-mag* bands. We estimate the missing magnitude values by using the closest filter that has the values of interest.
- We use the image for the *i-mag* as the detection frame and the image for the *z-mag* as the measurement frame to compute the magnitude value for *z-mag*. We use the image for *H-mag* as the detection frame, and *K-Mag* as the measurement frame to compute the magnitude for *K-Mag*, and use the image for *J-mag* as the detection frame for both *Z-mag* and *Y-Mag* measurements frames to compute magnitude values for *Z-mag* and *Y-Mag* respectively.
- Using data lineage queries, we locate the frames for all filters on which these sources were extracted or the frames where we expect the source to appear (dropout cases). This process can be done recursively up to the raw data as taken from the telescope.
- If reprocessing is required, the cutouts of the science frames together with the cutouts of the calibration frames for the specific regions on the

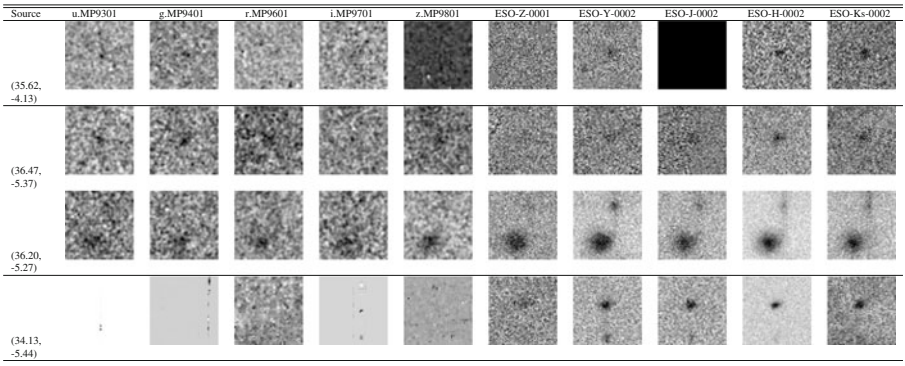


Fig. 13 The cutouts of the sources shown in Tables 2 and 3

Table 4 Performance parameters for extracting sources using double-image mode on full images

Project	Image size		File retrievals		
	Detection image (GB)	Measurement image (GB)	Detection image (s)	Measurement image (s)	Processing time (s)
VIKING	2	2	70.57	70.02	4525.30
CHFT	1.7	1.7	55.60	55.65	2145.31

science frames can be made and pushed through the sub-image-processing pipeline.
– For this case, re-processing is not required, what is needed is extracting one source from the measurement frame using parameters from the detection frame.
– For each source, the detection and measurements cutouts are submitted to the Astro-WISE sub-image-pipeline for source extraction.

The results are summarized in Table 3. The cutouts for the selected sources are shown in Fig. 13

7.2 Performance

The experiments described in this section were carried out on a dedicated Intel dual core 2.8Ghz desktop, with 4GB of memory, running GNU/Linux 2.6.18-238.12.1.e15, connected on 1Gbit network connection to the data server and database. However, the speeds of the test are heavily determined by the speed at which the database will send data to the local workstation for processing. For this reason, we show the time it takes to retrieve the files to the local node for processing. Table 4 shows processing details for the full images and Table 5 shows the processing details for sub-image processing. Notice that it took only 176.58 s on *VIKING* data to extract the information as required using sub-image processing as opposed to 4,525 s if we were to use full images.

One common characteristic of all dataflow programming frameworks is the requirement of locally staged data for processing. A computation cannot start before all required inputs are available locally at the processing node because data has to be transferred from archives to processing nodes. This is one of the performance bottlenecks in such systems. From the results in Tables 4 and 5, it is evident that a lot of time is saved when smaller images are transmitted from the storage nodes to the processing nodes over the network for processing.

Table 5 Performance parameters for extracting sources using double-image mode on a sub-images

Project	Image size		File retrievals		
	Detection image (KB)	Measurement image (KB)	Detection image (s)	Measurement image (s)	Processing time (s)
VIKING	20	20	0.12	0.18	176.58
CHFT	20	20	0.13	0.17	170.30

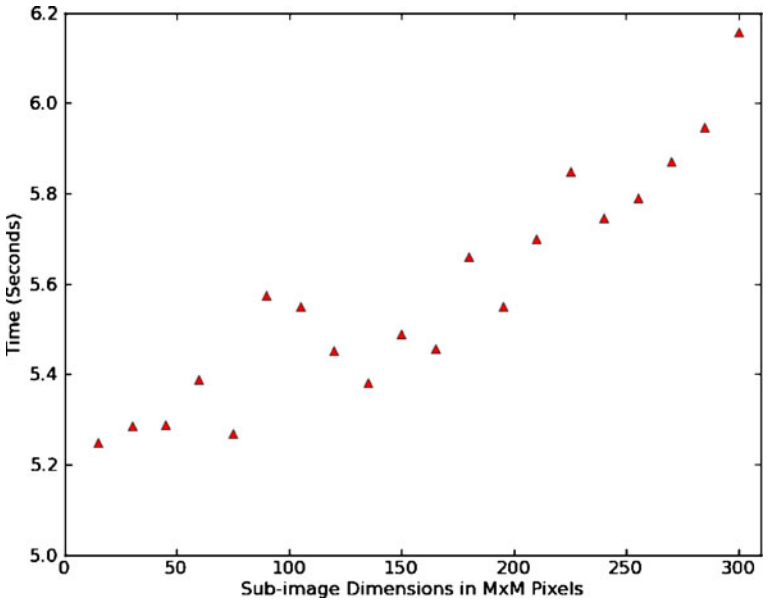


Fig. 14 A plot showing a relationship between the sub-image size and the time it takes to process a sub-image

We proceed to make and process cutouts of different sizes and we record the time it takes to process and extract sources from these cutouts. The time it took to process each different sized cutout was recorded. The results of this processing are shown in Fig. 14. Notice from the Fig. 14 that the processing time increases as the size of the cutout increases. This therefore shows that there is significant saving in computation time (and resources) while processing sub-images compared to processing full-images.

8 Related work

Most of the popular astronomical software languages and packages (Interactive Data Language,[9] European Southern Observatory Munich Image Data Analysis System[10] [14], Image Reduction and Analysis Facility [12]) assume a standalone approach to working with data. In the case of all these packages, the storage and access of the raw and processed data is a responsibility of the user. However, the astronomical data processing community (or the scientific community in general) is becoming very sophisticated. Data reduction and analysis has become complex [29], data rates have kept pace with advances in

[9] http://www.ittvis.com/language/en-us/productsservices/idl.aspx
[10] http://www.eso.org/sci/data-processing/software/esomidas

processing power (doubling roughly every two years), and the dimensionality of data is increasing [26]. As a result, several scalability issues have arisen which range from ensuring good performance, to handling large amounts of data, to capturing data lineage, and providing interfaces to interact with a large number of archives. Distributed or grid systems [30] have been developed to address performance and dataset concerns. Such systems like [17], provide a scalable infrastructure for running image pipelines in a distributed way.

Provenance-aware scientific workflow systems [10] have been considered as the paradigm for representing and managing complex distributed scientific computations. Systems such as those surveyed in [19] and [4] have enabled scientists to carry out complex scientific computations while capturing data lineage. Despite these developments, little or no support exists in current systems to trace lineage and pixel level. Most data lineage models do not allow end-users to use lineage data in scientifically meaningful way, in particular to improve scientific processes. Our lineage model introduced in this paper does capture lineage at the finest detail (e.g., a pixel transformation process). This lineage, specifically at pixel level is then used for various scientific processes, mainly in sub-image processing as described in this paper.

While tracing lineage at pixel level is missing out in most workflow systems, the level of granularity at which lineage is collected is linked to the particular needs of data lineage requirements. Different data lineage models are in use today and also several workflow management systems do exploit data lineage for different purposes. The use of lineage we describe in this paper is analogous to how other authors have used lineage to solve some use-cases. For example in [2, 23] and [18] data lineage has been used for results verification and to prove robustness of methods, in Kepler [1] data lineage has been used to enable smart "reruns" and process simplification of previously executed workflows and in [22] and [7], data lineage has been used for interactive design of workflows.

In this work, we extracted and processed cutouts from astronomical images according to the VO standards described in [27]. To the best of our knowledge, there haven't been attempts at sub-image processing reported in the literature. Therefore, we have no comparative analysis to evaluate if there could be advantages or disadvantages of our approach against any other approach. However, we show that in this current data deluge sub-image processing as compared to full-image processing supports research in distributed communities by transferring and processing a minimal set of data.

9 Discussion and conclusions

We have described a new framework that leverages data lineage to aid in selective retrieval and processing of data. We argue that sub-image processing is a powerful method that will provide efficient solutions for what are otherwise manual, time-consuming tasks.

This method provides scalable and easy-to-use primitives for reprocessing of data. With this kind of processing, even users at remote research centres could comfortably run and process data, without the limitations of huge data transfers and insufficient resources on local clients. We have proposed efficient algorithms and intuitive interfaces for realizing these primitives in Astro-WISE.

However, our approach is not foolproof. There are cases where it may fail to produce the results a user expects. For example, if a user applies the method to a processing that involves neighbouring pixels to determine a result of a pixel (e.g., derivation of astronomical parameters solution) the pipelines is likely to fail. However, when such a processing fails, or produces poor results, the user can initially process the full image and extract all parameters/needed to aid in processing of other sub-images. The effect shall be slower performance for the start, which improves significantly while processing other sub-images.

There are many avenues for future work. We intend to further transform scientific systems to process pixels rather than images. We are currently investigating how we can use databases to aid in such processing. One way that we are currently evaluating involves the loading of data into a database and then performing all processing inside the database, allowing accurate data lineage that can be easily captured and traced.

Acknowledgements Partially based on raw observations obtained with MegaPrime/MegaCam, a joint project of CFHT and CEA/DAPNIA, at the Canada-France-Hawaii Telescope (CFHT) which is operated by the National Research Council (NRC) of Canada, the Institut National des Science de l'Univers of the Centre National de la Recherche Scientifique (CNRS) of France, and the University of Hawaii. The data is processed within Astro-WISE system.

References

1. Altintas, I., Ludaescher, B., Klasky, S., Vouk, M.A.: Introduction to scientific workflow management and the kepler system. In: SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, p. 205. ACM, New York (2006)
2. Anderson, E.W., Ahrens, J.P., Heitmann, K., Habib, S., Silva, C.T.: Provenance in comparative analysis: a study in cosmology. Comput. Sci. Eng. **10**(3), 30–37 (2008). doi:10.1109/MCSE.2008.80
3. Begeman, K.G., Belikov, A.N., Boxhoorn, D.R., Dijkstra, F., Valentijn, E.A., Vriend, W.J., Zhao, Z.: Merging grid technologies. J. Grid Computing **8**, 199–221 (2010)
4. Bose, R., Frew, J.: Lineage retrieval for scientific data processing: a survey. ACM Comput. Surv. **37**(1), 1–28 (2005)
5. Calabretta, M.R., Greisen, E.W.: Representations of celestial coordinates in fits. Astron. Astrophys. **395**(3), 1077–1122 (2002)
6. Consortium, A.W.: Dependency cutout web interface. http://cutout.astro-wise.org/. Accessed June 2011
7. Ellkvist, T., Koop, D., Anderson, E.W., Freire, J., Silva, C.: Using provenance to support real-time collaborative design of workflows. Provenance and Annotation of Data and Processes: 2nd International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, 17–18 June 2008. Revised Selected Papers pp. 266–279 (2008). doi:10.1007/978-3-540-89965-5_27

8. Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for computational tasks: a survey. Comput. Sci. Eng. **10**(3), 11–21 (2008)
9. Fruchter, A.S., Hook, R.N.: Linear reconstruction of the hubble deep field (1996). http://www-int.stsci.edu/~fruchter/dither/drizzle.html. Accessed July 2011
10. Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L., Myers, J.: Examining the challenges of scientific workflows. Computer **40**(12), 24–32 (2007)
11. Glavic, B., Dittrich, K.R.: Data provenance: a categorization of existing approaches. In: BTW '07: 12. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web, pp. 227–241. Verlagshaus Mainz, Aachen (2007). ISBN 978-3-88579-197-3
12. Greenfield, P.: Reaching for the stars with python. Comput. Sci. Eng. **9**(3), 38–40 (2007). doi:10.1109/MCSE.2007.62
13. Greisen, E.W., Calabretta, M.R., Valdes, F.G., Allen, S.L.: Representations of spectral coordinates in fits. Astron. Astrophys. **446**(2), 747–771 (2006). doi:10.1051/0004-6361:20053818
14. Heck, A. (ed.): Information Handling in Astronomy—Historical Vistas. Springer-Verlag (2002)
15. Jaan, K.: Numerical Methods in Engineering with Python. Cambridge University Press (2005)
16. Mark, C.A., Francisco, V.G., Eric, G.W., Steven, A.L.: Representations of distortions in fits world coordinate systems. Astronomical Data Analysis Software and Systems XIII **314**, 551 (2003)
17. McFarland, J., Verdoes-Kleijn, G., Sikkema, G., Helmich, E., Boxhoorn, D., Valentijn, E.: The astro-wise optical image pipeline. Exp. Astron. (2010). doi:10.1007/s10686-011-9266-x
18. Miles, S., Wong, S.C., Fang, W., Groth, P., Zauner, K.P., Moreau, L.: Provenance-based validation of e-science experiments. Web Semant. **5**, 28–38 (2007). doi:10.1016/j.websem.2006.11.003. http://portal.acm.org/citation.cfm?id=1229184.1229197
19. Moreau, L.: The foundations for provenance on the web. Found. Trends Web Sci. **2**, 99–241 (2010). doi:10.1561/1800000010
20. Moreau, L., Ludäscher, B., Altintas, I., Barga, R.S., Bowers, S., Callahan, S., Chin, J.G., Clifford, B., Cohen, S., Cohen-Boulakia, S., Davidson, S., Deelman, E., Digiampietri, L., Foster, I., Freire, J., Frew, J., Futrelle, J., Gibson, T., Gil, Y., Goble, C., Golbeck, J., Groth, P., Holland, D.A., Jiang, S., Kim, J., Koop, D., Krenek, A., McPhillips, T., Mehta, G., Miles, S., Metzger, D., Munroe, S., Myers, J., Plale, B., Podhorszki, N., Ratnakar, V., Santos, E., Scheidegger, C., Schuchardt, K., Seltzer, M., Simmhan, Y.L., Silva, C., Slaughter, P., Stephan, E., Stevens, R., Turi, D., Vo, H., Wilde, M., Zhao, J., Zhao, Y.: Special issue: the first provenance challenge. Concurr. Comput.: Pract. Exper. **20**(5), 409–418 (2008)
21. Mwebaze, J., Boxhoorn, D., Valentijn, E.: Tracing and using data lineage for pipeline processing in astro-wise. Exp. Astron. (2011). doi:10.1007/s10686-011-9276-8
22. Scheidegger, C., Vo, H., Koop, D., Freire, J., Silva, C.: Querying and creating visualizations by analogy. IEEE Trans. Vis. Comput. Graphics **13**(6), 1560–1567 (2007). doi:10.1109/TVCG.2007.70584
23. Simmhan, Y., Plale, B., Gannon, D.: Karma2: Provenance management for data-driven workflows. Int. J. Web Service Res. **5**(2), 1–22 (2008)
24. Stephen, G.: The cfht legacy survey: stacked images and catalogs (2011). arXiv:1101.1084v2(astro-ph.CO).
25. Stevens, R., Zhao, J., Goble, C.: Using provenance to manage knowledge of In Silico experiments. Brief Bioinform. **8**(3), 183–194 (2007)
26. Szalay, A.S.: The sloan digital sky survey and beyond. SIGMOD Rec. **37**(2), 61–66 (2008). doi:10.1145/1379387.1379407
27. Tody, D., Plante, R.: Simple image access specification version 1.0 (2009). http://www.ivoa.net/Documents/SIA/. Accessed October 2011
28. Valentijn, E.A., McFarland, J., Snigula, J., Begeman, K., Boxhoorn, D., Renegelink, R., Helmich, E., Heraudeau, P., Kleijn, G.V., Vermeij, R., Vriend, W.J., Tempelaar, M.J.: Astro-wise: chaining to the Universe. In: Astronomical Data Analysis Software and Systems XVI, ASP Conference Series, vol. 376 (2007)
29. Wang, F., Luo, J., Deng, H., Liang, B., Ji, K.: C-swf: A lightweight scientific workflow system for astronomical data processing. International Workshop Computer Science and Engineering **2**, 64–67 (2009). doi:10.1109/WCSE.2009.767
30. Yu, J., Buyya, R.: A taxonomy of scientific workflow systems for grid computing. SIGMOD Rec. **34**(3), 44–49 (2005). doi:10.1145/1084805.1084814