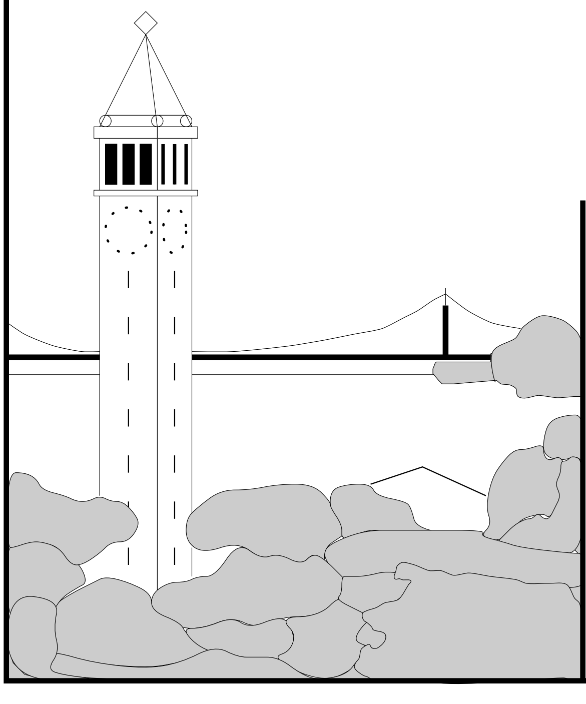
Supporting Fine-Grained Data Lineage in a Database Visualization Environment

A. Woodruff and M. Stonebraker



Report No. UCB/CSD-97-932

January 1997

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Supporting Fine-Grained Data Lineage in a Database Visualization Environment

Allison Woodruff and Michael Stonebraker
Department of EECS
University of California
Berkeley, CA 94720-1776*

Abstract

The lineage of a datum records its processing history. Because such information can be used to trace the source of anomalies and errors in processed data sets, it is valuable to users for a variety of applications including investigation of anomalies and debugging. Traditional data lineage approaches rely on metadata. However, metadata does not scale well to fine-grained lineage, especially in large data sets. For example, it is not feasible to store all of the information necessary to trace from a specific floating point value in a processed data set to a particular satellite image pixel in a source data set.

In this paper, we propose a novel method to support fine-grained data lineage. Rather than relying on metadata, our approach lazily computes lineage using a limited amount of information about the processing operators and the base data. We introduce the notions of weak inversion and verification. While our system does not perfectly invert the data, it uses weak inversion and verification to provide a number of guarantees about the lineage it generates. We propose a design for the implementation of weak inversion and verification in an object-relational database management system.

1. Introduction

Suppose a scientist applies a series of processing steps to an atmospheric data set and then views the result, a plot of cyclone tracks, in a database visualization system. The scientist sees an anomaly and wants to identify the input data which contributed to the unexpected value. The database system may be able to trace the lineage of the anomaly at a coarse level, using metadata. However, tracing from a specific cyclone track point in the processed data set to a particular array element in the source data set is not feasible using such an approach; the amount of metadata required would be far too large.

This type of scenario is common in the computational sciences. In this paper, we present an approach for deriving the lineage of a datum dynamically and at a fine granularity. Instead of relying on metadata, our approach combines a limited amount of knowledge about the processing operators with analysis of the base data. The approach works best when integrated into a database server using user-defined functions, but can be implemented outside of a database environment as well. In this paper, we describe our approach in terms of abstract properties and then in terms of implementation.

In general, the **lineage** of a datum 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. Many applications of lineage become evident if one considers processing history as a dataflow graph. For example, lineage information allows the user to trace the impact of faulty source data or buggy programs on derived data sets. It also allows the user to investigate the source data or programs that produced an anomalous data set. Such investigations may be difficult or impossible without data lineage: the user may not be familiar with processing steps which were written by an expert programmer. Further, tracing back through a number of processing steps is tedious and time-consuming, particularly if large data sets are involved.

The perceived importance of data lineage has grown in step with the increased volume and widened dissemination of processed data sets. The amount of support for data lineage has grown as well. For example, new scientific data standards (*e.g.*, the Spatial Data Transfer Standard [9], the Spatial Archive and Interchange Format [13], and the draft Content Standard for Digital Spatial Metadata [5]) gener-

*This work was sponsored by NSF under grants IRI-9400773 and IRI-9411334.

ally incorporate some kind of support for lineage. Recent scientific workflow systems (*e.g.*, GIS databases such as Geolineus [6] and geophysical databases such as BigSur [3]) automate the process of lineage tracking by providing direct support in the workflow infrastructure. BigSur, for example, can supply the entire graph of processing steps by which a given satellite image was produced, or a list of all images produced using a given processing step.

Thus far, research and development in data lineage has assumed that the complete history of a datum can and will be stored as a piece of metadata. A metadata-based approach to data lineage assumes that relatively coarse-grained information will suffice. For example, Earth scientists can easily afford to store, say, 100 bytes of metadata for the 100 6000x6000 raster images they receive in a day. Even outside of the database field, (*e.g.*, in such areas as dataflow program debugging [15]), researchers have assumed that the space/time cost of data tagging will be acceptable.

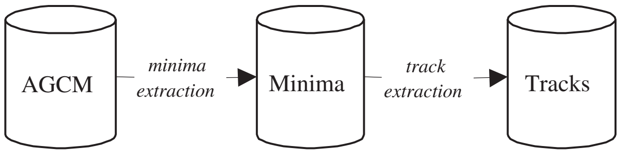
However, some scientific applications require lineage at a much finer granularity than previously considered - we know of applications that require lineage at the level of pixels in images [2]. Imagine a scientist who is debugging an application that uses a regridded (interpolated) composite of many raster images. This user might need to know which pixels of which original raster images were used in the construction of the composite. Because of the nature of the source data as well as the nature of the image processing and regridding algorithms, we may not be able to normalize much of the data dependency information. Can we reasonably afford, say, 60 bytes of metadata per pixel?

This paper describes an approach for supporting such fine-grained data lineage. Our approach uses a limited amount of information about the processing steps to infer the necessary lineage information on a lazy basis. In this way, we avoid computing and storing such information in advance. Our fine-grained data lineage technique therefore complements coarse-grained metadata techniques.

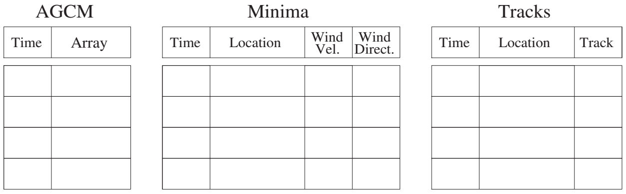
In the absence of explicit lineage, the most obvious way to identify relevant inputs is to invert the processing steps. A function f is said to be invertible if there exists some function f^{-1} such that for each element a input to f, $f^{-1}(f(a)) = a$. Unfortunately, only a limited number of functions are invertible.

We introduce the notion of **weak inversion**, which applies to a larger class of functions. Each function which is weakly invertible has a corresponding function f^{-w}. f^{-w} attempts to map from the output of f to the input of f, but is not guaranteed to be perfectly accurate. Instead, the accuracy of f^{-w} is described by a number of weaker guarantees. We also introduce the notion of **verification**. Verification functions refine the set identified by f^{-w}.

We propose the implementation of weak inversion and verification as extensions to an object-relational database

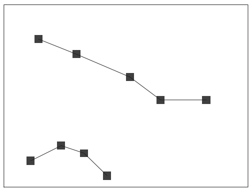


(a) Dataflow of cyclone track extraction.



(b) Schema of cyclone track extraction.

(c) SLP plot (single timestep).



(d) Cyclone track plot (multiple timesteps).

Figure 1. Cyclone track extraction.

management system (DBMS). We assume that users register their data processing functions in the DBMS and that the DBMS manages the application of these functions. Users register weak inversion and verification functions in the DBMS as well. Given a specific datum to invert, an inversion planner process infers which weak inversion and verification functions must be invoked, constructs a plan (function ordering), and then executes the plan by calling the corresponding sequence of functions within the DBMS.

In the remainder of this section, we present a sample scenario which is used to illustrate our principles throughout the paper. Section 2 defines our abstract model of weak inversion and verification. Section 3 describes how this model can be extended to a database environment and details the process by which expert users may register weak inversion and verification functions in an extensible database. Section 4 describes the inversion planner. Section 5 discusses our current status and future work. Finally, Section 6 presents conclusions.

Example scenario

As a real-world application of our techniques, we consider a scenario for extracting cyclone tracks from atmospheric simulation data, based on [8]. In this subsection, we present the processing steps used in this scenario. Throughout the paper, to illustrate portions of the model, we discuss the weak inversion and verification of functions in this example.

Validation of atmospheric simulations involves the comparison of model data to observational data. Cyclone tracks form one type of reference for such comparisons. The cyclone track extraction process begins with data generated by an Atmospheric General Circulation Model (AGCM). Two functions are applied to that data. The first function extracts local minima in sea level pressure (SLP), each of which may be the center of a cyclone. The second function assigns these minima to cyclone tracks. In the remainder of this subsection, we describe in detail the two processing steps (Figure 1a) and the schema of the data (Figure 1b).

The AGCM data consists of a series of multi-dimensional arrays, each with a time stamp. Each array is indexed by location and contains a variety of data about atmospheric conditions at a given location, *e.g.*, SLP value, wind velocity, and wind direction (see Figure 1c).

A feature extraction algorithm is applied to this data to locate minima. It is a neighborhood algorithm (described in more detail in Section 3.1.2) which outputs the following data about each minimum (shown in the Minima table in Figure 1b): time, location, wind velocity, and wind direction.

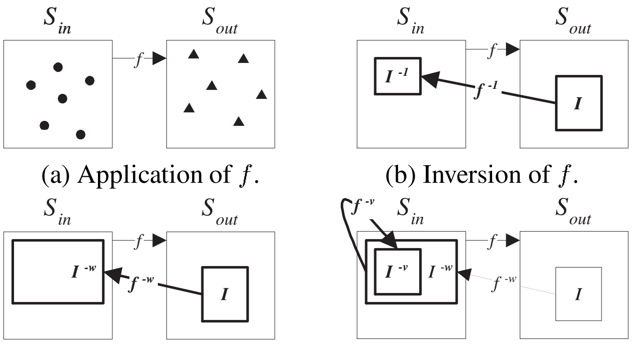
Cyclone track identification is performed on the Minima table. The track identification algorithm attempts to assign each minimum to the trajectory of some cyclone. To be assigned to a given track a, the minimum at time t must (1) have a certain proximity to the minimum in track a at the previous timestep $t-1$ or (2) be consistent with the wind velocity and direction of such a minimum. Some minima are not in fact cyclone centers and are not assigned to any cyclone track. The output of the cyclone track identification phase is a list of the time, location, and track number of the minima which were successfully assigned to tracks.

At this point, the user views the results of the data processing (see Figure 1d). They may be puzzled by one of the cyclone tracks because it does not match their expectations or agree with the observational data. They would like to select the apparently anomalous track and see all inputs which contributed to it. □

2. Abstract model

We begin with a function f which maps from a domain D to a range R. Now suppose the existence of sets $S_{in} \subseteq D$ and $S_{out} \subseteq R$ such that $f(S_{in}) = S_{out}$ (see Figure 2a). We are interested in inverting some subset of S_{out}. We call this subset I (the **image** of f).

If f is invertible, there exists some function $f^{-1} : R \rightarrow D$ such that $f^{-1}(f(A)) = A$ for any $A \subseteq D$. We say that $I^{-1} = f^{-1}(I) = \{x \in S_{in} | f(x) \in I\}$ is the **inverse image** of I, *i.e.*, the relevant members of S_{in} which map onto I (see Figure 2b).



(a) Application of f. (b) Inversion of f.

(c) Weak inversion of f. (d) Verification of f.

Figure 2. Weak inversion and verification.

Because not all functions are invertible, we are interested in a weak inverse function $f^{-w} : R \rightarrow D$.[1] $f^{-w}(I) = \mathcal{F}^{-w}$ is a set of values in the domain D (f^{-w} generates $\mathcal{F}^{-w}$ without reference to S_{in}). To find the members of $\mathcal{F}^{-w}$ which actually appear in S_{in}, we intersect the two sets. The result is called I^{-w}, *i.e.*, $I^{-w} = S_{in} \cap \mathcal{F}^{-w}$. I^{-w} approximates I^{-1} (see Figure 2c[2]).[3]

As an example, suppose f is a function which maps from integers to their squares and that I is the singleton set $\{9\}$ in S_{out}. f^{-w} can conclude without examining any portion of S_{in} that $\mathcal{F}^{-w} = \{3, -3\}$. If S_{in} contains 3 but not -3, $I^{-w} = \{3\}$.

Because f^{-w} does not find the perfect inverse, it is not guaranteed that $f^{-w}(f(I^{-1})) = I^{-1}$. Similarly, it is not guaranteed that $I^{-w} = I^{-1}$. Instead, we specify the relationship between I^{-w} (the set which is identified) and I^{-1} (the set which is actually of interest) with the following properties:

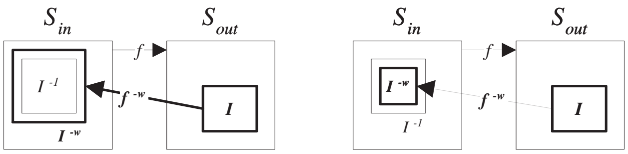
- complete: $I^{-w} \supseteq I^{-1}$. I^{-w} is complete in that it contains all items of interest. In this case, there are no false negatives, *i.e.*, f^{-w} does not exclude any items of I^{-1} from I^{-w} (see Figure 3a).
- pure: $I^{-w} \subseteq I^{-1}$. I^{-w} is pure in that it contains only items from I^{-1}. In this case, there are no false positives, *i.e.*, f^{-w} does not include any items in I^{-w} which are not in I^{-1} (see Figure 3b).[4]

[1] f^{-w} is, strictly speaking, a relation rather than a function. To avoid confusion with database relations, we refer to f^{-w} as a function.

[2] Recall that f^{-w} does not produce I^{-w} directly; rather, I^{-w} results from the intersection of $\mathcal{F}^{-w}$ (the output of f^{-w}) and S_{in}.

[3] We use the notation I^{-1} to describe the set of inputs which maps onto I even if f is not invertible and no f^{-1} exists.

[4] Complete and pure are related to the traditional information retrieval metrics **recall** (the fraction of relevant documents which are retrieved) and **precision** (the fraction of documents which are retrieved which are relevant) [4]. Specifically, complete represents perfect recall and pure represents perfect precision.



(a) Complete. (b) Pure.

Figure 3. Properties of weak and verified inverses.

Note that if $\mathcal{F}^{-w}$ is both complete and pure, $\mathcal{F}^{-w} = I^{-1}$, *i.e.*, $f^{-w}(I)$ is I^{-1}.

In general, we use the term **closeness** to describe the relationship between the weak inverse and I^{-1}. Note that not all sets with a given property are equivalent. For example, if two weak inverses A and B are complete, with $|A| < |B|$, A is closer because it contains fewer irrelevant items and therefore more closely approximates the actual inverse. Similarly, if two non-equal weak inverses are pure, the larger one is closer because it contains more relevant items (again, more closely approximating the actual inverse).[5]

We next observe that, because there exist functions that cannot be inverted without reference to the input values, not all functions have useful f^{-w}s (a useful f^{-w} outputs $\mathcal{F}^{-w} \neq D$). Therefore, we introduce a verification function, f^{-v}, which has access to the values in S_{in}. $f^{-v} : R \times D \rightarrow D$ takes I^{-w} and I as input and outputs a set $I^{-v} \subseteq I^{-w}$ (see Figure 2d). I^{-v} can be described by the same properties as I^{-w}, *i.e.*, I^{-v} can be complete or pure. In addition, f^{-v} can require that the input set I^{-w} be complete or pure. We term such restrictions **apply conditions**.

Our basic notation is summarized in Figure 4. Additionally, when necessary for our discussion, we use accents to distinguish particular functions or images, *e.g.*, f, $\hat{f}$, and $\tilde{f}$.

3. Concrete model

This section applies the concepts of the abstract model to our environment. Weak inversion and verification are being implemented in the Tioga database visualizer [11][1]. Tioga adopts the boxes-and-arrows programming paradigm popularized by AVS [14], Data Explorer [7], and Khoros [10]. Every box is a user-defined function and arrows represent the flow of data between these functions. Certain boxes are database browsers which visualize data and display it to the user. Tioga functions are written by expert users and registered in POSTGRES, an object-relational DBMS [12]. We are extending this registration mechanism so that the expert user can register weak inversion and verification functions.

In this section, we show how the set entities of the abstract model presented in Section 2 map onto database tuples and attributes. The fact that tuples have multiple attributes complicates the definition and application of the weak inversion and verification functions; we extend our model to address this issue. We then extend our model to allow the weak inversion and verification of portions of attributes, *e.g.*, an element in an array. Next, we extend the model from the inversion of single functions to the inversion of arbitrary dataflow graphs. Finally, we present the procedure the expert follows to register weak inversion and verification functions in POSTGRES.

3.1. Extending the abstract model to a database environment

Each function in Tioga takes as input some table T_{in} and yields as output some table T_{out}. These tables correspond to the input and output sets S_{in} and S_{out} of our abstract model. In this subsection, we discuss the inversion of attributes as well as the inversion of elements within complex attributes.

3.1.1 Attributes

We begin with the image I which is to be weakly inverted. I in general consists of a set of tuples, *i.e.*, a restriction of T_{out}. In our discussion, we consider that I consists of a single tuple. However, it is a straightforward generalization to consider images which contain multiple tuples or are the result of applying a restriction to T_{out}.

We have chosen to support weak inversion and verification at attribute-level granularity. This has two primary advantages over tuple-level granularity. First, the user is only required to provide weak inversion and verification functions for attributes in which they are interested or which they understand. Second, it allows more precise inversion.

The inverse image I^{-1} consists of the tuples in T_{in} which contain attributes which affected I. There is a separate weak inversion and verification process for each attribute within T_{out}. Therefore, f^{-w} for a tuple is comprised of a number of functions $f_1^{-w}...f_n^{-w}$. Each f_k^{-w} weakly inverts a specific attribute k of T_{out} (see Figure 5a). We define I_k as the projection of I on k. $\mathcal{F}_k^{-w} = f_k^{-w}(I_k)$ describes a subset of the domain which might have contributed to I_k. $\mathcal{F}_k^{-w}$ is a Boolean expression containing restrictions on T_{in}. I_k^{-w} is the result of $\mathcal{F}_k^{-w}$ applied to T_{in}. I_k^{-w} may be complete or pure with respect to I_k. Additionally, I_k^{-w} may possess a user-defined property with respect to I_k.

Similarly, f^{-v} for a tuple is comprised of a number of functions $f_1^{-v}...f_n^{-v}$. Each f_k^{-v} takes two inputs and verifies

[5] A possible general formulation of closeness involves a unit-cost similarity metric: $|A \cap I^{-1}| - |A \backslash I^{-1}| < |B \cap I^{-1}| - |B \backslash I^{-1}|$. However, for the rest of this paper, we only consider the case in which we compare two pure or two complete sets.

	Relationship to other sets	Description	Definitions
f	$f : D \to R$	Function applied to S_{in} to yield S_{out}.	
S_{in}	$S_{in} \subseteq D$	Input set.	
S_{out}	$S_{out} \subseteq R$	Output set.	$S_{out} = f(S_{in})$
I	$I \subseteq S_{out} \subseteq R$	Portion of S_{out} being queried (the image).	
I^{-1}	$I^{-1} \subseteq S_{in} \subseteq D$	Inverse image of I.	
f^{-w}	$f^{-w} : R \to D$	Weak inversion function of f.	
$\mathcal{F}^{-w}$	$\mathcal{F}^{-w} \subseteq D$	Filter on S_{in}.	$\mathcal{F}^{-w} = f^{-w}(I)$
I^{-w}	$I^{-w} \subseteq S_{in} \subseteq D$	Weak inverse of I.	$I^{-w} = S_{in} \cap \mathcal{F}^{-w}$
f^{-v}	$f^{-v} : R \times D \to D$	Verification function of f.	
I^{-v}	$I^{-v} \subseteq I^{-w} \subseteq S_{in} \subseteq D$	The verified inverse of I.	$I^{-v} = f^{-v}(I, I^{-w})$

Figure 4. Definitions.

a specific attribute k of T_{out}. The first input to an f_k^{-v} is I_k. The second input is the weak inverse I_k^{-w}. The user may register requirements (apply conditions) for the weak inverse, *i.e.*, an f_k^{-v} may require that an input I_k^{-w} possess a specific property or properties with respect to I_k. The output of an f_k^{-v} is I_k^{-v}; I_k^{-v} can be described by the properties we have previously defined.

The expert user writes and registers each f_k^{-w} and f_k^{-v} individually. The user may register zero or more weak inversion or verification functions for each attribute. If the user does not provide a weak inversion or verification function for a given attribute, the system provides a trivial default function (discussed below). Multiple weak inversion or verification functions for a given attribute may be desirable when different weak inversion or verification functions for an attribute have different properties. For example, one can imagine registering one f_k^{-w} which is complete but not pure and another which is pure but not complete.

Although the weak and verified inverses consist of entire tuples in T_{in}, the user may wish to know which attributes in T_{in} are related to some specific attribute in T_{out}. (We assume the existence of an interface through which the user specifies the attribute(s) in T_{out} of interest.) Such information can be inferred using the registration tables which are described in Section 3.2 and presented to the user.

Weak and verified inverses of the same attribute may be combined. Further, the weak and verified inverses of different attributes in the image can be combined to find the weak and verified inverses of the entire image. The resulting weak and verified inverses of the entire image can be described as having a given property or properties. In Section 4 we discuss methods of combining weak and verified inverses and the properties which result from these combinations. We assume that a small amount of bookkeeping is done during the combination of weak and verified inverses to preserve information about which attributes in T_{in} are related to which attributes in T_{out}.

As an additional complication, attributes in an image may be the result of either **aggregate** or **scalar** functions. As an example of an aggregate function, if an attribute a in T_{out} is the maximum value of some attribute in T_{in}, f_a is aggregate. However, if an attribute a in T_{out} is derived from values in exactly one tuple in T_{in}, f_a is scalar. In Section 4, we discuss the different combination methods which pertain to these two types of functions.

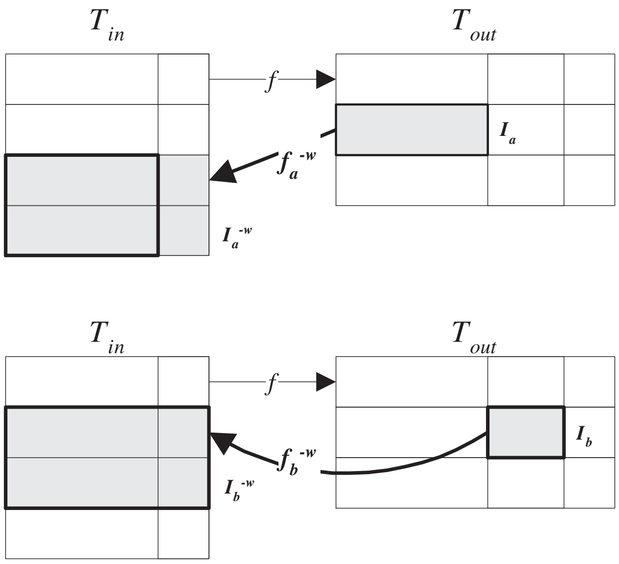
Example of weak inversion and verification of an attribute

We now return to our cyclone extraction scenario. In Figure 1b, we can write a trivial weak inversion function for the time field in Minima. Specifically, if I_{Time} consists of the single value t (recalling that we are assuming for purposes of this discussion that I is a singleton set), $\mathcal{F}_{Time}^{-w}$ is "`select * from AGCM where AGCM.Time = t`". □

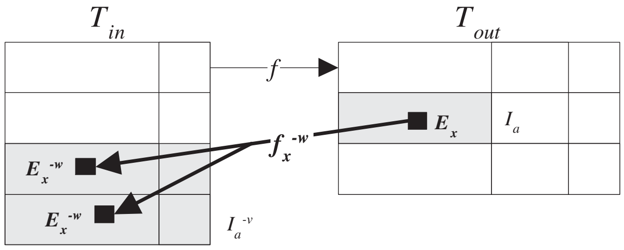
3.1.2 Complex Attributes and Elements

We have assumed above that I consists of simple attributes within tuples in T_{out}. However, POSTGRES supports a variety of complex attributes, *e.g.*, arrays, tuple types (in which an attribute may be broken down into a number of other attributes), and user-defined types (which can only be manipulated using the methods defined for the type). Observe that any attribute, whether simple or complex, can be weakly inverted and verified within the model described in Section 3.1.1.

We define an **element** to be a member contained in a complex attribute, *e.g.*, a cell within an array or an attribute in an instance of a tuple type. The user may wish to query an element within a complex attribute in T_{out}. For example, a



(a) Weak inversion of attributes a and b in I.



(b) Weak inversion of an element E in attribute a.

Figure 5. Weak inversion of multiple levels.

scientist may wish to invert a specific pixel within a satellite image. In this subsection, we extend our definition of weak inversion and verification functions to operate on elements within complex attributes.[6] Therefore, the user may register an f_k^{-w} for any dimension k in an array. Similarly, the user may register an f_k^{-w} for any attribute k of a tuple type. In either case, appropriate f_k^{-v}s may also be registered. (Note that each element of a complex attribute may in turn be a complex attribute. Therefore, weak inversion and verification functions may exist for an element within an element.) We assume the existence of some interface through which the user can specify some element which they wish to invert.

For example, suppose there exist tables T_{in} and T_{out}, each containing an attribute of the array type. Now imagine that E is an element within a specific array attribute a and that we wish to invert dimension x in a. Weak inversion and verification functions are applied to identify I_a^{-v} in T_{in}. Then, f_x^{-w}s are applied to each member of I_a^{-v}.[7] The result is E_x^{-w} (see Figure 5b).[8]

Weak inversion and verification functions are not required to return values at the same **level** as their arguments. (We use the term level to describe the degree of containment of an attribute or element. The top (first) level consists of the attributes in a given table, the second level consists of the attributes or dimensions contained by the top level, etc.). For example, the weak inversion of an element might yield a simple attribute. Similarly, the weak inversion of an attribute might yield an element.

Example of weak inversion and verification in the presence of complex attributes

Suppose we wish to identify the inverse of a specific minimum I in Minima in our cyclone track extraction scenario. The location attribute Minima.Location is directly related to the location index of the array in AGCM.Array. Recall that the function which extracts minima from AGCM.Array is a neighborhood algorithm. Minima at a location (x, y) are identified if they meet one of two criteria:

1. All immediate neighbors of (x, y) have a higher SLP than SLP(x, y).
2. The average SLP of the 5x5 neighborhood centered at (x, y) (but exclusive of (x, y)) is higher than SLP (x, y).

Therefore, the classification of I as a minimum may have resulted either from values of its immediate neighbors or from values of the 5x5 neighborhood surrounding it. Since there is no way to distinguish between these two cases without examining the values in the AGCM table, the weak inversion of $I_{Location}$ returns all cells in the 5x5 neighborhood centered at $I_{Location}$. This weak inverse is complete but not pure. The verification function has an apply condition which specifies that the weak inverse must be complete (it must be able to examine the entire 5x5 neighborhood).

The verification function examines the contents of the 5x5 neighborhood and determines which criteria applied. If the first applied, the verified inverse consists of the immediate neighborhood. If the second applied, the verified inverse consists of the 5x5 neighborhood. In both cases, the verified inverse is both complete and pure with respect to $I_{Location}$. □

3.1.3 Dataflow Graphs

Thus far, the model has not addressed multiple inputs or outputs to functions. However, a general dataflow graph is

[6] We do not currently support these operations for subparts of arbitrary user-defined types which by nature do not have accessors known to the database.

[7] The specific process is discussed in more detail in Section 4.2 below.

[8] We use I and E in this example to distinguish between the attribute and the element within the attribute. However, in general, we still consider the output of an f_k^{-w} (f_k^{-v}) to be I_k^{-w} (I_k^{-v}).

Information	*Type*
name of f_k^{-w}	string
name of f	string
nature of f_k	aggregate or scalar
image type	type of attribute or dimension k within T_{out} being weakly inverted
inverse image types	types of attributes or dimensions within T_{in} which app ear in I_k^{-w}
properties of output I_k^{-w}	complete and/or pure and/or user-defined

Figure 6. Information to register for weak inversion functions.

Information	*Type*
name of f_k^{-v}	string
name of f	string
nature of f_k	aggregate or scalar
image type	type of attribute or dimension k within T_{out} being verified
inverse image types	types of attributes or dimensions within T_{in} which appear in I_k^{-v}
properties of output I_k^{-v}	complete and/or pure and/or user-defined
apply conditions for I_k^{-w}	complete and/or pure and/or user-defined

Figure 7. Information to register for verification functions.

a DAG. We address this issue by restructuring the dataflow graph into groups of functions with one input and one output. We invert these groups separately. We then combine the results of the inversions.

More specifically, we define a **chain** in a dataflow graph to be a linear sequence of functions from an input to an output. Each function in the chain is called a **step**. An arbitrary dataflow graph may be broken down into a number of such chains. Each chain is inverted separately (the specific process for inverting a chain is discussed in Section 4). The results of the inversions of each chain are unioned to find the inversion of the entire dataflow graph.

3.2. Registration procedure

The expert user must register several pieces of information about weak inversion and verification functions. This information is used by the inversion planner described in Section 4 to infer which functions should be used for weak inversion and verification.

The user begins by identifying the name of the function which will perform the weak inversion or verification. The user next identifies the function f which is being weakly inverted and verified. The user also specifies whether the attribute being inverted results from an aggregate or scalar function, *i.e.*, f_k is described as aggregate or scalar.

The user must also register information which allows the inversion planner to infer which weak inversion and verification functions apply to a given attribute. Therefore, for each inversion function, the user specifies the types of the relevant attributes (or dimensions) in the image and in the inverse image. The inversion planner searches for attributes (or dimensions) in T_{in} and T_{out} which match these specified types.[9]

Finally, the user enters information about the properties of the sets output by the weak inversion and verification functions. Note that if a property is specified for an output set, it is guaranteed to be true. However, if it is not specified, it might or might not pertain. The information the expert user enters to register to register an f_k^{-w} or an f_k^{-v} is summarized in Figure 6 and Figure 6.

As mentioned in Section 3.1.1, in addition to the f_k^{-w}s or f_k^{-v}s registered by the user, every attribute or element resulting from every function has a default f_k^{-w} and a default f_k^{-v}. The default f_k^{-w} outputs a filter consisting of no

[9] This typing system may lead to ambiguities, *e.g.*, if two attributes of the same type appear in T_{in} and the registered information for f does not permit us to infer which function produces which attribute. In an alternative design, the user might specify the precise names of the attributes in the image and inverse image. However, such a design limits the degree to which weak inversion and verification functions may be easily reused, forcing the user to explicitly register weak inversion and verification functions for every attribute which is to be inverted.

restrictions, *i.e.*, $I_k^{-w} = T_{in}$. The default I_k^{-w} is therefore guaranteed to be complete, but it is not guaranteed to be pure or to possess any user-defined properties. The default f_k^{-v} outputs I_k^{-w}. Therefore, the default I_k^{-v} has precisely the same properties as the I_k^{-w} it takes as input. Note that if both defaults are used, $I_k^{-w} = I_k^{-v} = T_{in}$.

User-defined properties are registered in a separate mechanism in which the user specifies the name of the property and the combination rules which apply to it (either complete or pure rules as described in Section 4.1).

4. Inversion planner

The inversion planner is responsible for devising a plan to weakly invert and verify the image selected by the user. The result of the execution of this plan must match the user's specification of properties as closely as possible (*e.g.*, the user may specify that they wish the verified inverse image to be complete or pure). The plan specifies which weak inversion and verification functions will be applied to which tables in what order.

In this section, we discuss how properties of weak and verified inverses can be preserved during the combination of weak inversion and verification functions. We then present the algorithm the inversion planner follows to invert a chain.

We make several simplifying assumptions. In Section sec:implementation we consider further optimizations which may be made if these assumptions are relaxed.

- We assume all tables (including intermediate results) are materialized.
- We assume there is a per-tuple cost of applying an f_k^{-w} or an f_k^{-v} (as opposed to a fixed or per-byte cost).
- We assume that the planner is trying to find the closest possible verified inverse.
- We assume that the desired properties as specified by the user are the same for all verified inverses in a chain.

4.1. Preservation of properties

We have already discussed several properties of sets (complete, pure, user-defined). Some combinations of sets preserve such properties and some do not. We begin our discussion of the preservation of properties by detailing inversion of a single function (simple attributes and complex attributes). Next, we discuss inversion of multiple functions.

4.1.1 Preservation of properties during the inversion of simple attributes

This subsection covers two primary types of combinations. First, for a given dimension k, I_k^{-w}s or I_k^{-v}s may be combined to improve the closeness of the inversion of k.[10] Second, to this point, we have only considered weak inversion and verification of attributes in the image. However, after all individual attributes have been fully inverted, a higher-level combination may also be performed. Specifically, the results of the inversion of multiple attributes may be combined to assemble the inverse of an entire tuple. (Similarly, the results of the inversion of multiple dimensions within a complex attribute may be combined to assemble the inverse of the complex attribute.) In the latter part of this subsection, we describe how such combinations may be advantageous.

We begin by considering the case in which T_{out} contains exactly one attribute, y. Recall that multiple weak inversion functions f_y^{-w} may be registered for a single attribute (*i.e.*, the inversion planner may have the choice of several weak inversion functions). We have already observed that it is desirable to have different weak inversion functions which have different properties. However, it is also desirable to have multiple weak inversion functions with the same property to increase the closeness of the weak inversion. There are two interesting cases. First, suppose we have two weak inversion functions, each of which returns a pure set (call them A and B).[11] If $A \neq B$, the union of A and B yields a strictly larger pure set (the larger a pure set, the more accurate it is). Second, if the weak inversion functions yield complete sets, the intersection of A and B yields a strictly smaller complete set (the smaller a complete set, the more accurate it is). Observe that these rules of combination apply whether f_y is scalar or aggregate. Figures 4.1.1a and 4.1.1b illustrate the combination of A and B in this case.

Now we consider the combination of the weak inverses of multiple dimensions. First, we discuss the case in which the f_ks are scalar. If a tuple in I^{-1} is relevant to a tuple in I, it must be relevant to each attribute of I, *i.e.*, for all k, it must be a member of I_k^{-1}. Therefore, in general, if multiple attributes are scalar, the weak inverses of these should be intersected to find the weak inverse of the entire tuple (or if multiple dimensions are contained in a complex attribute, the intersection of the weak inverses of each dimension finds the weak inverse of that complex attribute).

As a concrete example, consider the case in which T_{out} contains precisely two attributes x and y. Suppose the image I consists of exactly one tuple which we wish to invert.

Now suppose that I_x^{-w} is complete in relation to x. (This

[10] In general, we use the term dimension to refer to either an attribute which is being weakly inverted and verified or a dimension within an array which is being weakly inverted or verified.

[11] Examples in this section refer to the combination of I_k^{-w}s. The same rules apply to $\mathcal{F}_k^{-w}$s and I_k^{-v}s.

does not guarantee that it is complete in relation to y.) Also assume that I_y^{-w} is complete in relation to y. (Again, this does not guarantee that it is complete in relation to x). We see that any tuple which is relevant to I must be relevant to both the x and y values in I; since both I_x^{-w} and I_y^{-w} are complete, all relevant tuples must appear in both sets. Therefore, $I_x^{-w} \cap I_y^{-w}$ contains all possible tuples which may be relevant to I, *i.e.*, $I_x^{-w} \cap I_y^{-w}$ is complete in terms of I. However, some of the tuples in the intersection may not be relevant. The combination of a complete I_x^{-w} and a complete I_y^{-w} is illustrated in Figure 4.1.1c.

Pure sets follow the same combination rules, although the logic is slightly different. Suppose that I_x^{-w} is pure in relation to x and I_y^{-w} is pure in relation to y. Any tuple which appears in both sets is guaranteed to be relevant to both x and y. The intersection therefore contains only relevant tuples, although it may not contain all relevant tuples.

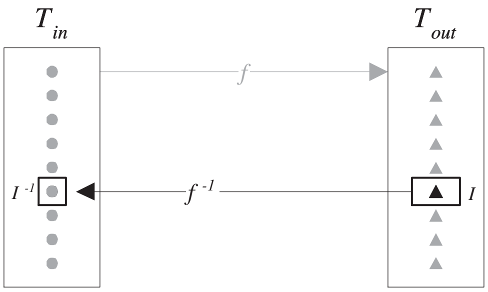
Next, we discuss the case in which f_ks are aggregate. Suppose T_{out} contains two attributes x and y. In this situation, there is no guarantee that a single tuple in T_{in} must be relevant to I. For example, suppose x is the maximum of an attribute a in T_{in} and y is the maximum of an attribute b in T_{in}. The weak inverses of x and y may be disjoint; however, both are relevant to I. Therefore, all weak inverses associated with aggregate f_ks (whether complete or pure) should be unioned.

Finally, we consider the case in which some f_ks are scalar and some are aggregate. All weak inverses associated with scalar f_ks should be intersected as specified. Then, all weak inverses associated with aggregate f_ks should be unioned. As the last step, both of the resulting sets should be unioned. Observe that whether we are combining sets within one attribute or combining sets for multiple attributes, there is no case in which it is desirable to combine a pure set with a complete set.

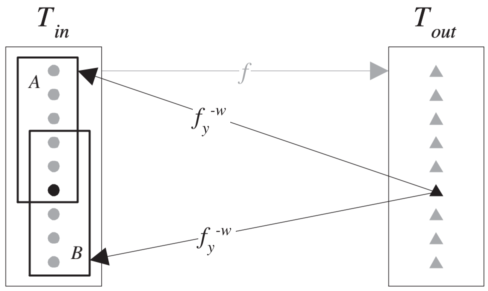
In some cases, we may invert a subset of the attributes in an image (either because the user has specified that only those attributes are of interest or because interesting weak inversion and verification functions are not available for all attributes). Notationally, if a set has certain properties with respect to multiple attributes 1...k in the image, we say that it has those properties with respect to $I_{1...k}$.

Example of weak inversion and verification of multiple scalar attributes

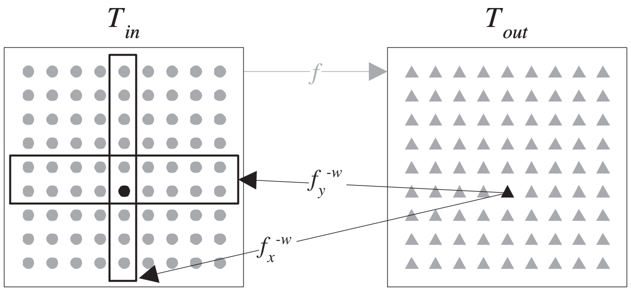
Returning to our cyclone track extraction example, consider the weak inversion and verification of an image in the Tracks table. We suppose that weak inversion and verification functions have been registered for the attributes Time and Location. Observe that f_{Time} and $f_{Location}$ are scalar. Also note that each of these attributes has trivial weak inversion and verification functions which yield sets which are complete and pure with respect to the individual attributes



(a) Inversion of an attribute.



(b) Combination of two complete I_y^{-w}s.



(c) Combination of a complete I_x^{-w} and a complete I_y^{-w}.

Figure 8. Combination of weak inverses.

Time and Location (the values in Minima are identical to those in Tracks). Therefore, since the forward functions are scalar, we intersect I^{-v}_{Time} and $I^{-v}_{Location}$. The result is complete and pure with respect to $I_{Time,\ Location}$. □

4.1.2 Preservation of properties during the inversion of complex attributes

Recall from Section 3.1.2 that the weak and verified inverses of an image can exist at multiple levels. In this subsection, we consider the properties of these different levels in the weak and verified inverses. First, we discuss the properties of a single level in the weak and verified inverses. Second, we discuss the properties of multiple levels in the weak and verified inverses.

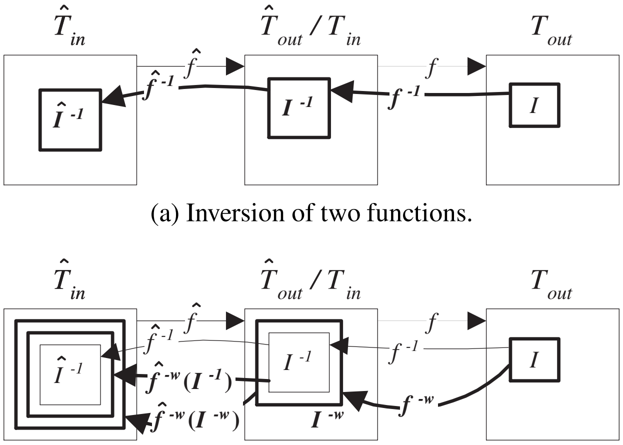
In Section 4.2, we present the specific process by which the weak and verified inverses are identified. For now, it is sufficient to understand that each level in the weak and verified inverses is calculated by a set of independent weak inversion and verification functions. The weak inversion and verification functions for a single level in the inverse may invert different levels in the image. For example, one function may weakly invert an attribute in the image to tuples in the weak inverse; another function may weakly invert an element in the image to tuples in the weak inverse as well. In general, the resulting weak or verified inverses are combined according to the combination rules described in Section 4.1.1 above.

Since each level in the inverse is computed separately, each level in the inverse can have different properties with respect to the various levels in the image. However, the properties of a level in the inverse affect the properties of all levels below it. Specifically, the weak or verified inverse of a lower level can only have a given property with respect to a level in the image if all higher levels in the inverse have that same property. This implies that levels in the inverse are computed in a top-down manner, which in turn implies that each level in the inverse is a subset of the higher levels.

For example, suppose that in Figure 5b, f^{-v}_a identifies tuples containing satellite images which contributed to an aggregate satellite image I_a and that the output of f^{-v}_a is complete with respect to I_a. Now suppose that f^{-w}_x identifies the region of each satellite image which contributed to E_x and that its output E^{-w}_x is pure with respect to E_x. Observe that f^{-w}_x might be applied to a member of I^{-v}_a which is not relevant to I_a; the resulting E^{-w}_x is therefore not relevant to I_a. Consequently, E^{-w}_x is not pure with respect to I_a (it would be pure only if I^{-v}_a were pure).

Example of preservation of properties during inversion of complex attributes

Consider the inversion of an image I in the Minimum table of the cyclone track extraction example. We perform



(a) Inversion of two functions.

(b) Weak inversion of two functions.

Figure 9. Weak inversion of a chain.

this inversion in two steps.

First, we weakly invert and verify the top level of AGCM. We assume weak inversion and verification functions are available for Minima.Time and Minima.Location. We apply weak inversion and verification functions to identify $\hat{I}^{-v}_{Time}$ (both $\hat{\mathcal{F}}^{-w}_{Time}$ and $\hat{f}^{-v}_{Time}$ restrict AGCM.Time). $\hat{I}^{-v}_{Time}$ is complete and pure. Applying the weak inversion function to $\hat{I}_{Location}$ yields a complete set $\hat{I}^{-w}_{Location}$ which consists of all tuples in AGCM. Applying the verification function to $\hat{I}^{-w}_{Location}$ yields a complete and pure set $\hat{I}^{-v}_{Location}$ which also consists of all tuples in AGCM (the verification function is able to verify that the array in each tuple contains $\hat{I}_{Location}$). At this point we use our combination rules: we intersect $\hat{I}^{-v}_{Time}$ and $\hat{I}^{-v}_{Location}$ to find a verified inverse $\hat{I}^{-v}$ which is complete and pure with respect to $\hat{I}_{Time,\ Location}$.

Next, we weakly invert and verify the second level of AGCM. We use the weak inversion function described in Section 3.1.2 to generate a filter $\tilde{\mathcal{F}}^{-w}_{Location}$. We apply this filter to every member of $\hat{I}^{-v}$. The result $\tilde{E}^{-w}_{Location}$ is complete with respect to $\hat{I}_{Location}$. We then apply the verification function $\tilde{f}^{-v}_{Location}$ to $\tilde{E}^{-w}_{Location}$, which yields a complete and pure set $\tilde{E}^{-v}_{Location}$. Since the weak inversion and verification of both levels is complete and pure, the result of the second level inversion is complete and pure with respect to $\hat{I}_{Time,\ Location}$. □

4.1.3 Preservation of properties during the inversion of multiple functions

In this section, we show how our abstract model generalizes to chains. We observe that the properties of weak and verified inverses are transitive.

For example, consider a chain with two functions $\hat{f}$ and f

in which the output of $\hat{f}$ is input to f. Suppose that an expert user has registered functions which provide weak inversion and verification of each of these functions.

Now suppose an end user wishes to find the inverse of an image in T_{out} (see Figure 9a). The user would like to identify the relevant inputs in both T_{in} and $\hat{T}_{in}$. Ideally, the system would use f^{-1} to invert the image and identify I^{-1} in T_{in}. Then, it would treat I^{-1} in T_{in} as an image in $\hat{T}_{out}$ and find its inverse $\hat{I}^{-1}$ in $\hat{T}_{in}$.

We apply our weak inversion functions in this situation as follows (we assume these weak inversion functions are complete). We begin by finding a weak inverse in T_{in}.[12] However, the user wishes to see the relevant inputs from $\hat{T}_{in}$ as well. This is accomplished by using I^{-w} as an image in $\hat{T}_{out}$. Recall that the weak inverse can differ from the actual inverse image. Chaining weak inversion functions together amplifies this inaccuracy. In Figure 9b, we see that $\hat{f}^{-w}(I^{-w})$ yields a larger (and more inaccurate) set than $\hat{f}^{-w}(I^{-1})$.

Despite this loss of accuracy, we can still make certain guarantees about the relationship of weak inverses to inverse images in these situations. The key observation is that complete and pure are both transitive properties. Specifically, if both f^{-w} and $\hat{f}^{-w}$ are complete, so are their outputs. In Figure 9b, therefore, both $\hat{f}^{-w}(I^{-1})$ and $\hat{f}^{-w}(I^{-w})$ are complete, though neither is pure.

Example of preservation of properties during inversion of multiple functions

Consider the dataflow diagram in Figure 1a. We saw in Section 4.1.1 that the weak inversion and verification of certain attributes in Tracks yields a complete and pure verified inverse in Minima. Similarly, we saw in Section 4.1.2 that the weak inversion and verification of certain attributes in Minima yields a complete and pure verified inverse in AGCM. Therefore, the weak inversion and verification of an image in Tracks yields complete and pure verified inverses in both Minima and AGCM. □

4.2. Inversion planner algorithm

In this subsection, we first discuss ordering constraints for weak inversion and verification. We then present an algorithm which follows these constraints as well as those presented in Section 4.1.

We have discussed several stages of weak inversion and verification of a chain. If weakly inverting or verifying some part of the chain impacts the weak inversion or verification of some other part, we say the former part **affects** the latter.

[12] It would also be possible to find a filter or a verified inverse and use it as the image. Each of these cases has different performance implications, as discussed in Section 5.1. For simplicity, we assume that we are using the weak inverse for the remainder of this section.

```
for each step num_steps to 1
  for each left_level 1 to num_left_levels
    for each right_level
      for each dimension k in 1 to num_attributes
        find all f_k^{-v}s which have the desired property
        find all f_k^{-w}s with matching apply conditions
        remove the f_k^{-v}s which can't be satisfied
      end;
    end;
  end;
  for each left_level 1 to num_left_levels
    for each right_level k in 1 to num_attributes
      for each dimension
        apply all f_k^{-w}s
        apply the filters to get the I_k^{-w}s
        combine the I_k^{-w}s for the dimension
      for each right_level
        apply all f_k^{-v}s
      combine the I_k^{-v}s within each dimension
    end;
    combine the I_k^{-v}s across dimensions
  end;
end;
```

Figure 10. Algorithm for inverting a chain.

For example, consider Figure 9b. If the weak inversion and verification of f were more accurate, then I^{-w} would be more accurate. The improved I^{-w} could be used as an image input to $\hat{f}^{-w}$ resulting in a more accurate $\hat{I}^{-w}$. Therefore, we say that the weak inversion and verification of f affects the weak inversion and verification of $\hat{f}$.

There are two critical observations about which parts of the chain can affect others:

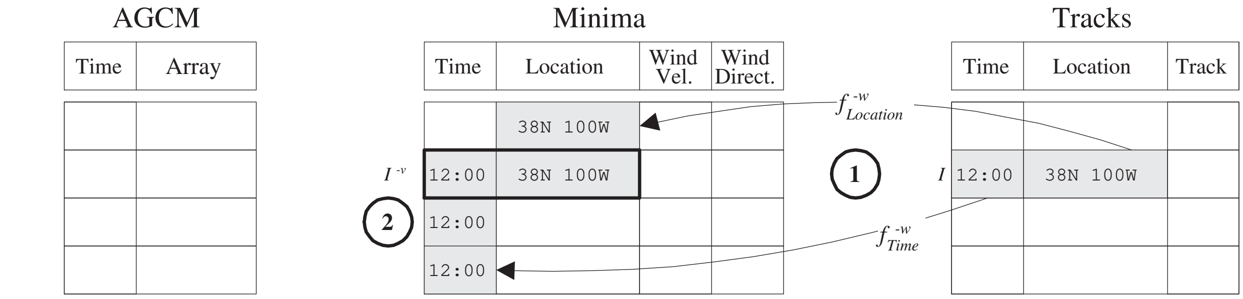
1. The inversion of a step can affect the inversion of any step to its left (although it can not affect the inversion of steps to its right).

2. Within a step, one inversion can affect another inversion which yields a lower level in the inverse (since the lower levels in the inverse are a subset of the higher levels). Conversely, one inversion can not affect another inversion which yields a higher level in the weak or verified inverse.

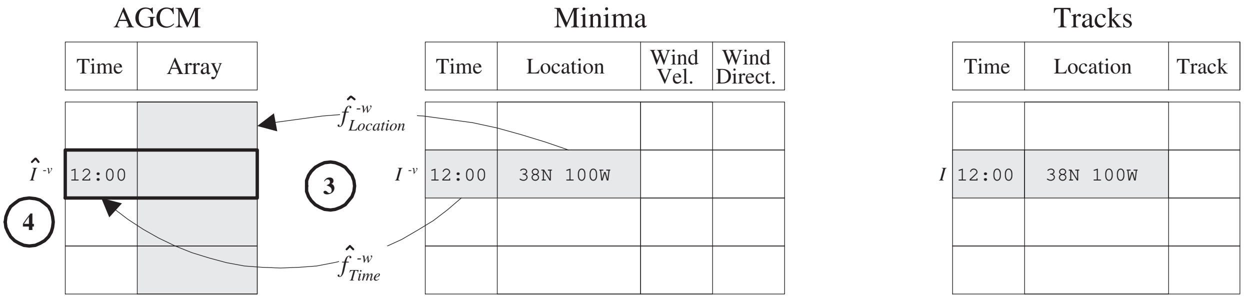
These observations suggest a natural ordering of the inversion process.

1. Steps should be weakly inverted and verified proceeding from right to left.

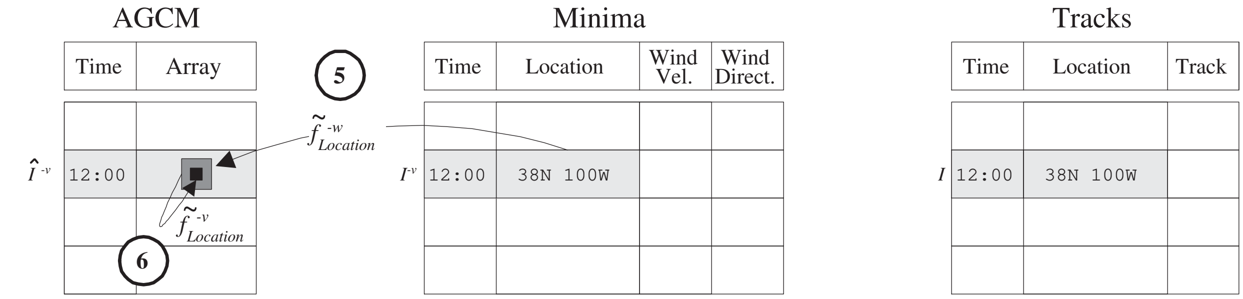
2. Within a step, each level in the inverse should be computed (weakly inverted, verified, and combined) before the levels below it are computed.



(a) Identifying the verified inverse of the track extraction.



(b) Identifying the top-level verified inverse of the minima extraction.



(c) Identifying the second-level verified inverse of the minima extraction.

Figure 11. Inversion of cyclone track extraction.

Combining the constraints of Section 4.1 (preservation of properties) and Section 4.2 (ordering) we arrive at the algorithm presented in Figure 10.

Example application of algorithm

We have discussed all the weak inversions and verifications necessary to trace the relevant inputs of the cyclone track extraction scenario presented in Section 1. According to the algorithm for the inversion planner, the complete weak inversion and verification of an image I in Tracks would consist of the following steps:

- Weakly invert and verify I, yielding a complete and pure verified inverse I^{-v} in Minima (as described in Sections 3.1.1 and 4.1.1). Specifically, the system will:

  1. Weakly invert and verify the attributes Tracks.Time and Tracks.Location.
  2. Intersect I^{-v}_{Time} and $I^{-v}_{Location}$ (see Figure 11a[13]).

- Weakly invert and verify I^{-v} yielding a complete and verified inverse $\hat{I}^{-v}$ in AGCM (as described in Sections 3.1.2 and 4.1.2). Specifically, the system will:

  3. Weakly invert and verify the attributes Min-

[13]For clarity, in Figure 11, we illustrate only attributes which are in the weak and verified inverses rather than the entire tuple. As mentioned in Section 3.1.1, a small amount of bookkeeping is done to facilitate such a presentation to the user.

ima.Time and Minima.Location, finding the top level of the verified inverse.

4. Intersect $\hat{I}^{-v}_{Time}$ and $\hat{I}^{-v}_{Location}$. The result is $\hat{I}^{-v}$ in AGCM. $\hat{I}^{-v}$ is the set of the members of AGCM which are relevant to I^{-v} (see Figure 11b).
5. Weakly invert the attribute Minima.Location, finding the second level of the weak inverse. The result is $\tilde{E}^{-w}_{Location}$ which is complete but not pure.
6. Verify $\tilde{E}^{-w}_{Location}$ yielding $\tilde{E}^{-v}_{Location}$ (see Figure 11c). $\tilde{E}^{-v}_{Location}$ is both complete and pure (as discussed in Section 4.1.3). □

5. Implementation and future work

We are in the process of implementing weak inversion and verification (as described in Section **??**) in the POSTGRES DBMS. We are also implementing a number of weak inversion and verification functions.

5.1. Efficient rematerialization of intermediate results

In this paper, we assumed that all intermediate results in a given chain of dataflow operators had been materialized. However, many dataflow systems cache their intermediate results (*e.g.*, Data Explorer [7]). Suppose the user of a caching system finds an anomaly and attempts to view an intermediate datum that contributed to it. In the worst case, the system must rematerialize all intermediate results simply to recreate one datum in one intermediate step.

Weak inversion enables us to recreate lost intermediate results efficiently. Previously, we invoked f^{-w}s on either user-specified attribute values of interest (*i.e.*, the original image) or attribute values that contributed to the computation of those interesting values (*i.e.*, the verified inverses in the intermediate results). However, we can also invoke f^{-w}s on the constant values found in our $\mathcal{F}^{-w}$ filters (which were generated by f^{-w}s operating on images to their right in the dataflow diagram). Using this fact, we can generate a sequence of $\mathcal{F}^{-w}$ filters all the way to the base table of our dataflow chain. We can then apply these $\mathcal{F}^{-w}$ filters as additional restrictions at each intermediate step in the chain, reducing the amount of data that must be processed.

5.2. Efficient materialization of partial results

The technique described above extends trivially to the efficient materialization of partial results. For example, if we allow the user to specify bounds on the (end result) regions to be visualized, the coordinates of that bounded region become another set of constants to which we can apply f^{-w}s. This implies that we can generate a chain of weak-inverse restrictions similar to that described in Section 5.1, again reducing the amount of data that must be processed.

5.3. Extended semantic query optimization

A related idea can be used for semantic query optimization in databases that support user-defined functions. Consider the following query:

```
SELECT name
FROM   Employees E
WHERE  tax(E.salary) > 30000
```

If we know that the lowest tax rate is 15constant through the call to the tax function and produce a direct filter on salary. This is useful if salary is indexed, or even if salary is not indexed but tax is expensive to compute. For example, we could transform the query above to read as follows:

```
SELECT name
FROM   Employees E
WHERE  tax(E.salary) > 30000
       AND E.salary > 200000
```

This technique can be applied to conventional functions and operators. In addition, it can be combined with techniques taken from previous work in semantic query optimization. For example, inferences can be made using join clauses that enable us to push down weakly inverted versions of one tables restrictions onto other tables.

5.4. Reuse of common subexpressions

The process of breaking dataflow graphs into linear components may produce components with shared steps. This is undesirable because I^{-w}s and I^{-v}s may be computed redundantly. However, the problem is not as simple as finding overlapping steps because we can only share inverses between two components if, for example, the inverses would result from applying the same f^{-w}. The problem is further complicated if components require different properties. If we allow the user to specify that one dataflow graph source must be pure and another must be complete, then overlapping components may require I^{-w}s and I^{-v}s that have been produced using different properties. Sharing then becomes more difficult, if not impossible.

6. Conclusions

We have proposed a method to support fine-grained data lineage. Rather than relying on metadata, our approach lazily computes lineage using a limited amount of information about the processing steps. This approach incorporates weak inversion and verification. While the system does not perfectly invert the data, it provides a number of guarantees about the lineage it generates on the fly.

We have proposed a design for the implementation of weak inversion and verification in an object-relational DBMS. This functionality has a number of interesting applications. We have discussed how it can help users track the lineage of specific data.

We are currently exploring several ways in which weak inversion and verification can be applied to optimization problems such as view maintenance, efficient materialization of partial results, and enhanced semantic query optimization.

Acknowledgements: We would like to thank Paul Aoki and Joe Hellerstein for comments on previous drafts of this paper. We would also like to thank Paul Aoki for valuable discussions of this work.

References

[1] A. Aiken, J. Chen, M. Stonebraker, and A. Woodruff. Tioga-2: A direct manipulation database visualization environment. In *Proc. 12th Int. Conf. on Data Engineering*, pages 208–17, New Orleans, LA, Feb. 1996.

[2] E. Bainto, J. Dozier, J. Frew, J. Gray, R. Mechoso, and S. Miley. Requirements for Sequoia database system. Sequoia 2000 technical memorandum, University of California, Berkeley, CA, Sept. 1993.

[3] P. Brown and M. Stonebraker. BigSur: A system for the management of Earth science data. In *Proc. 21st Int. Conf. on Very Large Data Bases*, pages 720–728, Zurich, Switzerland, Sept. 1995.

[4] C. Cleverdon and M. Keen. *Factors Determining the Performance of Indexing Systems*. ASLIB Cranfield Research Project, 1966.

[5] Federal Geographic Data Committee. Content standards for digital spatial metadata (final draft), June 1994.

[6] D. P. Lanter. Design of a lineage-based meta-data base for GIS. *Cartography and Geographic Information Systems*, 18(4):255–261, 1991.

[7] B. Lucas, G. Abram, N. Collins, D. Epstein, et al. An architecture for a scientific visualization system. In *Proc. 1992 IEEE Visualization Conference*, pages 107–114, Boston, MA, Oct. 1992.

[8] E. Mesrobian, R. Muntz, J. Santos, E. Shek, C. Mechoso, J. Farrara, and P. Stolorz. Extracting spatio-temporal patterns from geoscience datasets. In *Proc. IEEE Workshop on Visualization and Machine Vision*, pages 92–103, Seattle, WA, June 1994.

[9] National Institute of Standards and Technology. Federal information processing standard (FIPS PUB) 173-1: Spatial data transfer standard (SDTS), 1994.

[10] J. Rasure and M. Young. An open environment for image processing software development. In *Proc. 1992 SPIE Symposium on Electronic Image Processing*, pages 300–310, San Jose, CA, Feb. 1992.

[11] M. Stonebraker, J. Chen, N. Nathan, C. Paxson, and J. Wu. Tioga: Providing data management for scientific visualization applications. In *Proc. 19th Int. Conf. on Very Large Data Bases*, pages 25–38, Dublin, Ireland, Aug. 1993.

[12] M. Stonebraker and G. Kemnitz. The POSTGRES next-generation database management system. *Communications of the ACM*, 4(10):78–92, Oct. 1991.

[13] Surveys and Resource Mapping Branch, Ministry of Environment, Lands and Parks. Spatial archive and interchange format (SAIF), release 3.1, 1994.

[14] C. Upson, T. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. VanDam. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):32–40, July 1989.

[15] H. Yamana, J. Kohdate, T. Tasue, and Y. Muraoka. An environment for dataflow program development of parallel processing system-Harray. *Systems and Computers in Japan*, 22(8):26–38, 1991.