Why not?

CONFERENCE PAPER · JANUARY 2009

Source: DBLP

2 AUTHORS, INCLUDING:

Adriane Chapman

MITRE

28 PUBLICATIONS 957 CITATIONS

Why Not?

Adriane Chapman
The MITRE Corporation
McLean, VA 22102
achapman@mitre.org

H.V. Jagadish
University of Michigan
Ann Arbor, MI 48109
jag@umich.edu

ABSTRACT

As humans, we have expectations for the results of any action, e.g. we expect at least one student to be returned when we query a university database for student records. When these expectations are not met, traditional database users often explore datasets via a series of slightly altered SQL queries. Yet most database access is via limited interfaces that deprive end users of the ability to alter their query in any way to garner better understanding of the dataset and result set. Users are unable to question why a particular data item is Not in the result set of a given query. In this work, we develop a model for answers to WHY NOT? queries. We show through a user study the usefulness of our answers, and describe two algorithms for finding the manipulation that discarded the data item of interest. Moreover, we work through two different methods for tracing the discarded data item that can be used with either algorithm. Using our algorithms, it is feasible for users to find the manipulation that excluded the data item of interest, and can eliminate the need for exhausting debugging.

Categories and Subject Descriptors

H.1.2 [**User/Machine Systems**]: Human information processing; H.2.8 [**Database Applications**]: Scientific databases; E.0 [**General**]: Data

General Terms

Algorithms, Human Factors, Verification

Keywords

Provenance, Lineage, User understanding, Result explanations

1. INTRODUCTION

Why did the chicken not cross the road? Why not Colbert for President? Why did Travelocity not show me the Drake Hotel as a lodging option in Chicago? Why did Frank Sinatra not have brown eyes? Except for the unfathomable chicken, there is an explicit reason for each of these events not occurring[1]. Understanding why events do not occur is a natural process we use to understand our world. In the arena of databases and software systems, these questions often sound like: Why did this program not complete? Why did this tuple not appear in the result set? etc. The typical response to such questions is an epic debugging session in which the exact series of events is painstakingly traced until the answer is found.

Provenance, or the history of a piece of data, has been studied in order to explain where data came from [5, 9, 10] and what happened to it along the way [1, 2, 12, 19]. This information can be utilized to assist us in understanding why data items exist within a result set [1, 17, 19]. Provenance in these works can help explain surprises within a result set. However, what happens when the surprise is not what is found within the result set, but what is missing from the result set? Consider the following set of user problems:

- A scientist searches a biological database for: "sterol AND organism='Homo sapiens"'. A known function of ABC1 is "sterol transporter activity", so why is it not in the result set?
- A business traveler searches for flights on a popular flight booking web site, he cannot understand why there is no direct flight from DTW to LAX listed. He took that flight last week, so why is it not in the result set?
- A fan wants to see all the scoop about "the king" in *Return of the King* and types "Vito Mortensen" in IMDB. No Vito Mortensen is returned. Why not?

There is one running theme throughout the problems encountered above, despite the differences in domain: the user does not have the ability to alter their query in any way to garner better understanding of the dataset and result set. For instance, in a standard database system, if the user queries: SELECT name FROM employees WHERE salary >$100,000, and there are no results, the natural inclination is to slightly alter the query. Thus, the user may turn around and enter: SELECT name FROM employees WHERE salary >$75,000. In other words, an experienced classic database user has the means to explore the database and query space. A traditional database user is comfortable using this methodology to explore the characteristics of a dataset, and would have no need to ask WHY NOT?. Unfortunately, many applications and users no longer fit this paradigm. In the above examples, the users are not database users, they are application users who have no access to the underlying dataset. They cannot sift through the dataset to determine WHY NOT? when they encounter an unexpectedly missing result. Additionally, the applications themselves limit the type of queries the users can submit. In the Business Traveler Example

SIGMOD'09, June 29–July 2, 2009, Providence, Rhode Island, USA.
Copyright 2009 ACM 978-1-60558-551-2/09/06 ...$5.00.

[1] On November 1, 2007, the South Carolina Democratic Party executive council refused Colbert's ballot application by a 13-3 vote. Graduate students don't make enough to stay at the Drake Hotel, as noted in my cost preferences. According to Mendelian Inheritance, Sinatra did not have the dominant gene required.

above, Travelocity only allows the user to choose dates and location; it is impossible for the user to subtly alter the query to comb through the dataset to find the flight he thinks he knows about. Finally, in a traditional database, a standard, well-understood set of operators exist. In many applications this is not true, and the presence of complex, programmatic operations will obfuscate why data is not in the result set. In the biology example, why is ABC1 not in the result set after the query? The user knows that there is a database behind the application, but how does the keyword query interface with it? How are the results retrieved and displayed? Is there a bug? In actuality, the only reason ABC1 is not in the result set after this query is because the biological database only displays the top 100 hits, and ABC1 falls outside the range. This sort of Why Not? question could never be addressed via the sift and comb database search method.

This work was inspired by biological scientists attempting to understand the results presented by bioinformatics tools. These users *know* everything about their favorite biological entity (DNA, RNA, protein, organism, etc). When an informatics tool presents results contradictory to the expectations engendered by this knowledge, there is no way for the scientists to pick apart the underlying data or computations. Typically, this leads the scientists to throw up their hands and walk away from the tool forever.

1.1 The Problem

For ease of explanation, we will leave the biological domain, and present examples using books. After performing a set of relational operators, application functions, or mixture of both, a result set is formed. For instance, the data found in *Ye Olde Booke Shoppe*, in Table 1, is the result set of a manual curation of Library A and a Natural Language Processing of Library B, with a merge and duplicate removal process applied to the two outputs. In other words, a set of non-relational manipulations created the result set. When a user queries the *Ye Olde Booke Shoppe* database, a set of relational operators, and perhaps user functions, is used.

Once a result set is formed, if a user is unable to find what she wished, she must specify what she is seeking, using key or attribute values. Using this information, we describe how to offer explanations to the user about why the data is not in the result set.

Example 1. *Table 1 contains the contents of Ye Olde Booke Shoppe. If a shopper knows that all "window display books" are around $20, and wishes to make a cheap purchase, she may issue the query: Show me all window-books. Suppose the result from this query is: (Euripides, "Medea"). Why is (Hrotsvit, "Basilius") not in the result set? Is it not a book in the book store? Does it cost more than $20? Is there a bug in the query-database interface such that her query was not correctly translated?*

Why Not? is a series of statements about the potential reasons the data of interest to the user is missing from the result set. We can leverage provenance records [4, 6, 13], query specification and the user's own question to help understand Why Not?. In the example above, we can trace (Hrotsvit, "Basilius")'s progress through all the manipulations performed on (Euripides, "Medea"). Every manipulation at which the two do not behave similarly is a possible answer to "Why Not?".

Throughout the rest of this work, for ease of reader comprehension, we utilize a classic book database, with standard relational operators, and a few user defined, "server-side" functions. However, we would like to emphasize that the problem we are addressing exists outside of traditional databases, and our techniques can be applied to applications as well.

Author	Title	Price	Publisher
	Epic of Gilgamesh	$150	Hesperus
Euripides	Medea	$16	Free Press
Homer	Iliad	$18	Penguin
Homer	Odyssey	$49	Vintage
Hrotsvit	Basilius	$20	Harper
Longfellow	Wreck of the Hesperus	$89	Penguin
Shakespeare	Coriolanus	$70	Penguin
Sophocles	Antigone	$48	Free Press
Virgil	Aeneid	$92	Vintage

Table 1: The set of books in *Ye Olde Booke Shoppe*.

In Section 2, we provide a model and definitions that allow us to describe a piece of data not in the result set, and ask why it is not there. Moreover, we provide a model which allows us to answer Why Not? questions. In Sections 3–4 we discuss how Why Not? answers can be computed. The evaluation of our methods is presented in Section 5. In Section 6, we discuss related work; we conclude in Section 7.

2. FOUNDATIONS

Throughout this work, we call the basic logical data unit a *data item*. Data items may be tuples in a relational table, elements in XML, objects of arbitrary granularity in an OODB, etc. One data item may completely include, overlap with, or be totally disjoint from another data item. A data item contains a set of *attributes*. A data item that is a tuple contains standard relational attributes; a data item that is an XML element contains attributes that are child elements or attributes. Each attribute is associated with a data value. Attributes can be single or multi-valued. A *dataset* is comprised of a set of data items.

Datasets are often manipulated via workflows such as [3, 6, 18]. A Manipulation is a basic unit of processing in a workflow or query evaluation plan. Each Manipulation takes one or more data sets as input and produces a dataset as output. We write $M(D^{I_1}, D^{I_2}, ...) = D^O$ to indicate that Manipulation M takes datasets D^{I_1}, D^{I_2}, etc as input to generate data set D^O as output.

For example, the Manipulation Select_Books_≤$20 applied to the *Ye Olde Booke Shoppe* (shown in Figure 1(a)) dataset produces an output set comprising (Euripides, "Medea"), (Homer, "Iliad"), and (Hrotsvit, "Basilius"). An instance of a Manipulation applied to a specific data item we call a *manipulation*. We write $m(d^{I_1}, d^{I_2}, ...) = d^O$, where $d^{I_1} \in D^{I_1}$, $d^O \in D^O$, etc. m is an instance of M applied to specific data items d^{I_x} within dataset D^{I_x}. For example, an instance of Apply_SeasonalCriteria, in Figure 1(a), applied to the book (Hrotsvit, "Basilius") might result in $\emptyset$.

In short, a Manipulation is a discrete component of a workflow, and uses a set of specific attributes from the input dataset. In our work, we are intentionally agnostic about the granularity of a Manipulation. If the entire "workflow" comprises a single complex database query, then each operator in the query tree may be treated as a Manipulation. When dealing with a more complex workflow in which the database query is only one piece, an entire relational query may be a single Manipulation. We could, in an application-dependent manner choose to be at any intermediate point between these, and may even have Manipulations at different granularities within the same application. Some Manipulations relevant to our running example are as follows:

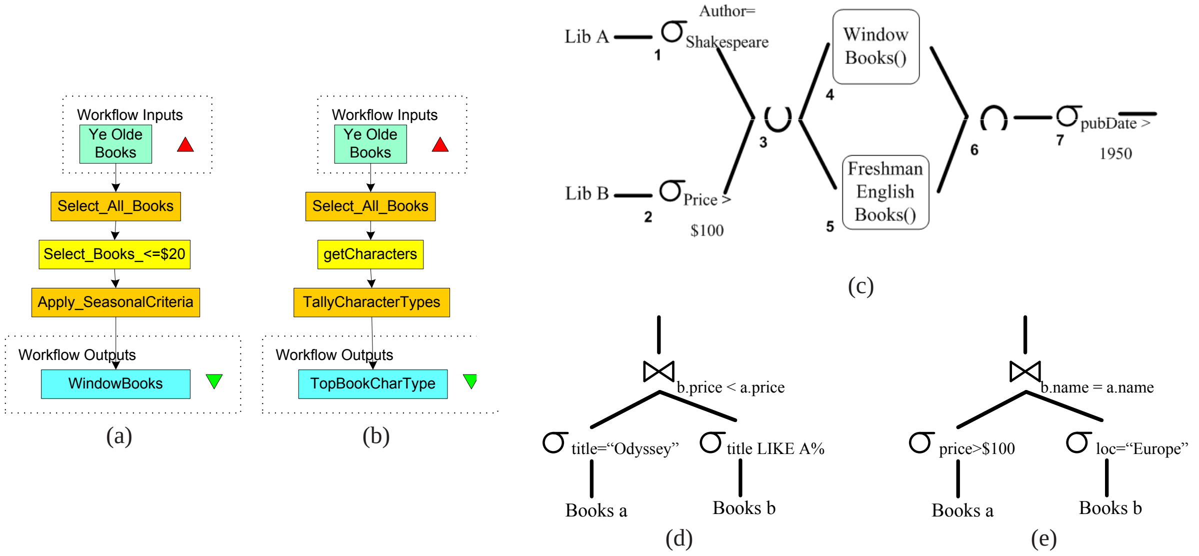


Figure 1: A set of workflows and query evaluation plans. (a) Finds the Window Display for *Ye Olde Booke Shoppe*. (b) Determines the top character genre in *Ye Olde Booke Shoppe*. (c) Creates a result set with all Shakespeare books in LibA and all books >$100 in LibB, determines the intersection of "Window Books" and "Freshman English Books" in this set and outputs any that were published after 1950. (Operators are numbered for ease of reference.) (d) Queries *Ye Olde Booke Shoppe* for all books priced less than The Odyssey. (e) Queries *Ye Olde Booke Shoppe* for all books priced greater than $100 and written in Europe.

MANIPULATION 1. $Selection$
Selects a subset of data from an input dataset, D, based on a selection condition on a data item's attribute.

EXAMPLE 2.
*SELECT * FROM books*
WHERE price <$70

MANIPULATION 2. $Apply\_SeasonalCriteria$
Returns all books that satisfy a set of seasonal criteria, which is defined as a black box function.

EXAMPLE 3.
Based on the date, Mother's Day is the next commercial holiday. The Seasonal Criteria black box, M, determines that Fillicide is a good seasonal topic. d^{I_1} is (Hrotsvit, "Basilius", $20); d^{I_2} is (Euripides, "Medea", $16); d^{I_3} is (Homer, "Iliad", $18). d^O is (Euripides, "Medea"), since "Medea" is the only book fits the seasonal criteria.

2.1 WHY NOT? Identity

When attempting to answer WHY NOT?, we have three known pieces from which to draw information: the query, the result set and the question. The query, Q, is the original query or workflow posed against a dataset D, and can be broken down into a series of MANIPULATIONS. The result set, R, is the result of that query on the dataset. We assume that the dataset D comprises a set of data *items*, not necessarily all of the same type. For example, if D is a set of relational tables, the data items may be individual tuples in these tables. Similarly R comprises a set of result items. A result item in R does not have to be in D – it could have been derived by composition of multiple items and other manipulations. We further assume that each item (in both D and R) has an associated set of attributes. If the item is a relational tuple, its attributes are the individual fields of the tuple.

The user asks a WHY NOT? question of the form "Why does R not contain any results satisfying predicate S." The predicate S is defined over (some subset of) the set of attributes A of D, using positive predicate logic over individual attributes. By this we mean that each atomic predicate is evaluated over a single attribute, and atomic predicates are combined using AND and OR, but without negation. While more complex predicates could be allowed in theory, our extensive analysis of user needs suggested that positive predicate logic was sufficient. A predicate that includes negation introduces a double negative into the statement of the WHY NOT? question, which makes it awkward to state in English and difficult to understand.

A data item d is said to *satisfy* predicate S, if the values of the attributes of d cause S to evaluate to TRUE. Note that this is possible even if d does not have all the attributes in S. For example if S is a disjunction of atomic predicates, it suffices if d defines one attribute and causes the corresponding atomic predicate to be TRUE. A weaker condition is that of *satisfaction-compatibility*. The predicate S can be described as a tree, with an atomic predicate at each leaf and an AND or an OR operator at each internal node. Given the values of attributes in d, some atomic predicates (at the leaves) will evaluate TRUE, others will evaluate FALSE, and yet others will remain undefined. Based on these, we can evaluate the internal nodes. If the root evaluates TRUE, the predicate is satisfied even if many internal and leaf nodes remain undefined. At any AND internal node of the tree, if there is no child evaluating FALSE and at least one child evaluating TRUE, then we can pretend that the undefined children are all also TRUE to determine compatibility.

If, with this change, the root evaluates TRUE, then we say that the data item d is *satisfaction-compatible* with predicate S, even if it does not in itself satisfy it.

To answer the user question, we must trace back from it, and understand the relationship between data items in the output to data items in the input. For this purpose, there is a well-accepted notion of *lineage* [5, 9, 10]. From [10], "Given a [manipulation] instance $\tau(I) = O$ 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, and we denote it as $I^* = \tau^*(o, I)$". In other words, the *lineage* of a data item is the set of input tuples that have influenced the inclusion or appearance of that data item in the result set. Using our running example of the query evaluation plan in Figure 1(d), and the result of (Sophocles, "Antigone"), lineage will pinpoint the exact tuples in the input set that contributed to (Sophocles, "Antigone") being in the result set. In this case, the lineage of (Sophocles, "Antigone") is (Sophocles, "Antigone") and (Homer, "Odyssey"). We utilize the definitions found in [5, 9, 10] for lineage with the following exception: the lineage of a MIN or MAX output data item is the data item(s) containing the reported value, not the entire input set. In this work, we denote this version of a lineage relationship with $I^* \overset{m}{\angle} o$, where m is a manipulation.

Using the concept of lineage, we can define the set of items relevant to the users WHY NOT? question as follows:

DEFINITION 1. *Unpicked:*
A data item $d \in D$ is said to be unpicked, *if*
i. There exists an attribute $a \in A$ that is both associated with d and appears in the user question predicate, S,
ii. d is satisfaction-compatible with S, and
iii. d is not in the lineage of any result item in R.

EXAMPLE 4.
Consider the shopper in Example 3. She may ask, "Why is "Basilius" not in the result set?". The Unpicked data item, (Hrotsvit, "Basilius"), is specified by its title attribute. Observe that the attribute of interest is not explicitly named, and the question predicate S not explicitly stated. We deduce the intended predicate based on an attribute value match. If the user had instead asked "Why are no 'Hesperus' books in the result set?", we would have found books with Publisher=Hesperus, "Epic of Gilgamesh", and Hesperus in the title field, "Wreck of the Hesperus." Of course it is also possible to specify a specific attribute-value pair, such as title="Coriolanus".

If the user instead asks, "Why are there not (more) books by Homer in the result?," we would find all data items with an attribute value of "Homer", but then only the Iliad is in the Unpicked, since the Odyssey is in (the lineage of) the result set. Alternatively, given the results (Sophocles, "Antigone") after the execution of the evaluation plan in Figure 1(d), the user may ask "Why not "Free Press" and "Penguin" books?". In this case, only "Penguin" will be used to identify Unpicked data items from the input set, since "Free Press" is in the lineage of the result data item.

Finally, note that every data item in D may be Unpicked if the user question is "Why is the result $\emptyset$" or "Why not anything".

The concept of *lineage* is central to much provenance work, and is the natural means to address WHY questions. We used this concept in our definition of Unpicked. Unfortunately, lineage is a concept that applies only to data items in the result set, tracing data items through manipulations from the result set to the input sets. In fact, lineage can be used to answer any WHY NOT? query in which the presumed Unpicked actually exists in the result set. However, the data items we are interested in are **NOT** in the result set, and therefore do not have lineage. Thus, we must define a new concept, *successor*, that will permit us to trace forward from the input rather than trace back from results.

DEFINITION 2. *Successor:*
Given a manipulation m that takes in dataset I and outputs O, $d' \in O$ is a successor of $d \in I$, iff $d \overset{m}{\angle} d'$.

Even though an Unpicked data item by definition does not exist in the result set, or even after a manipulation, we can use this definition of successor to watch how Unpicked data items move through workflows. Notice that a successor depends purely upon the notion of lineage, not attribute values. After a query, if a user asks, "Why not $61?", it does not matter if a manipulation projects out the attribute $61. Using lineage, the tuple (Sophocles, "Antigone") is directly associated with the input tuple (Sophocles, "Antigone", $61). In other words, attribute preservation is not required.

3. WHY NOT? ANSWERS

DEFINITION 3. *Picky Manipulation:*
A manipulation m is "Picky" with respect to an Unpicked data item u if:
i. u or a successor of u is in the input set of m, AND
ii. there is no successor of u in the output set of m.

For example, consider the evaluation plan in Figure 1(d), and the Unpicked data item (Virgil, "Aeneid"). Looking only at the $\sigma_{title=''Odyssey''}$ manipulation, the Unpicked data item is in the manipulations input set, and its Successor is not in the output set. Thus $\sigma_{title=''Odyssey''}$ is a Picky Manipulation for this data item. In other words, a picky manipulation is one that causes an unpicked data item to be excluded from the output set of a manipulation.

Given a set of Unpicked data items, we can identify one or more picky manipulations for each. The union of all these manipulations is the set of picky manipulations. We can present these to the user in response to the WHY NOT? question. However, this set of manipulations may often be large. Users may be overwhelmed with dozens of manipulations, each of which filtered something out. While this is technically a correct and complete answer, in most situations users are likely to want to know only about the highest level (in the query tree) or latest (in workflow) picky manipulations. These are closest to the final result and hence "most visible" to the user. If desired, the user can dig deeper from here. With this in mind, we define:

DEFINITION 4. *Frontier Picky Manipulation:*
A manipulation is "Frontier Picky" with respect to an Unpicked data item set U if:
i. the manipulation is Picky for at least some $u \in U$, and
ii there does not exist $u \in U$ for which a successor of u occcurs later in the workflow.

Continuing with the example of why (Virgil, "Aeneid") is not in the result set after the workflow in Figure 1(d), even though $\sigma_{title=''Odyssey''}$ is a Picky Manipulation, it is not a Frontier Picky Manipulation since an Unpicked Successor exists going into $\bowtie_{b.price<a.price}$. Thus, $\bowtie_{b.price<a.price}$ is not just a Picky Manipulation, but since no Unpicked Successors exist later in the workflow, it is the Frontier Picky Manipulation.

Notice that whether a MANIPULATION is picky or not is dependent upon the Unpicked data item of interest. In the above example, if the user wonders why (Virgil, "Aeneid") is not in the result set, the Frontier Picky Manipulation is $\bowtie_{b.price<a.price}$. However, if

the user wonders why (Hrotsvit, “Basilius”) is not in the result set, the Frontier Picky Manipulation is $\sigma_{titleLIKEA\%}$.

The discussion and examples thus far have focused on a singular path of MANIPULATIONs. However, this does not need to be the case. The execution of a workflow is a directed acyclic graph (DAG), and can thus have many paths, as in Figure 1(c).

Let us walk through the series of operations in Figure 1(c), following the data item (Euripides, “Medea”). Operators **2**, **5** and **7** are potentially Picky Manipulations. If (Euripides, “Medea”) were fed into any of these operators, it would not be part of the output. However, manipulation **7** is not picky since the successors of (Euripides, “Medea”) never reach it. Operator **5** is the Frontier Picky Manipulation. Because the intermediate results further down the DAG still contain (Euripides, “Medea”) despite **2** not including it as a successor in the intermediate result set; **2** is Picky, but not Frontier Picky[2].

This leads to a formulation of what can be used to answer a user’s WHY NOT? question.

DEFINITION 5. WHY NOT? *Answer:*
Given a user question regarding why not predicate S in a result set R produced by workflow W, comprising manipulations M, upon an input data set D, the answer to the user question comprises the set of frontier picky manipulations in M with respect to the items in D identified as unpicked according to S and W.

In other words, a WHY NOT? answer will return the manipulation(s) $m \in M$ at which the last Unpicked successor was excluded from the result set.

For instance, consider the query evaluation plan in Figure 1(e), which finds all books whose authors are from Europe and are priced greater than $100. Given the input dataset in *Ye Olde Booke Shoppe* and a result set of ∅, a user may ask “why were no results returned” (a.k.a. Why not anything?). If results are produced from both the selection on the both the books and author table, the join will be called the Frontier Picky Manipulation, even though both selections may themselves also be Picky with respect to particular Unpicked items. If the query instead were slightly altered, to seek books priced greater than $1000, then the price selection would be the frontier picky manipulation, since there are no books costing more. The join no longer has any input and so is no longer Picky. However, the other selection remains Picky for some items even in this case. If we further modify the query to seek books whose authors are from Antarctica and are priced greater than $1000, then both selections are identified as Frontier Picky. Notice that the set of Frontier Picky manipulations produce the answers one would intuitively expect. In the original query, the response is that there is no book that satisfies both requirements simultaneously, though there are bools that satisfy each individually. For the second query, the response is that there are no books priced over $1000 (though there are books by European authors). For the third query, the response is that no books are priced over $1000 and no books are by Antarctic authors.

3.1 Determining WHY NOT?

The definitions above immediately lead to a simple Bottom Up evaluation strategy described below. An alternative, Top Down, stategy can also be developed, which can turn out to be significantly more efficient depending on the position of the Frontier Picky Manipulation(s).

3.1.1 Bottom Up

A generic bottom up algorithm to find the Frontier Picky Manipulation, and thus the answer to WHY NOT?, is presented in Algorithm 1. It checks the output of every manipulation beginning at the DAG sources and proceeding in topologically sorted order. Whenever it finds no Unpicked successors at the output of a manipulation, it has found a Picky Manipulation. To find the Frontier Picky Manipulation, we have to continue through the DAG and make sure that Unpicked successors do not appear later in the DAG, for example through an alternate, disjuntive, path.

Algorithm 1: Answering WHY NOT? Bottom Up.

```
Input: DAG, $M$, of manipulations, $m$
Input: Global Input Dataset, $D$
Input: Queue, $Q$, initialized with Source
Input: Unpicked, $U$
Output: Frontier Picky Manipulation(s), picky
# Run in Breadth First Search order from Source to Sink
forall $m$ manipulations $\in$ Queue $Q$ do
    $O_m$ = output set of $m$, computed by evaluating $m$ on its inputs;
    $U_m$ = Unpicked inputs of $m$;
    if successorExists($O_m$, $U_m$) then
        forall $c$ manipulations $\in$ $m$.children do
            Record $O_m$ as an input to $c$;
            $c$.numparent- -;
            if $c$.numparent==0 then
                $Q$.add($c$);
            end
        end
    end
    else
        flagPossPicky($m$);
    end
end
forall $m$ manipulations flaggedPossPicky do
    forall $n$ manipulations flaggedPossPicky do
        if $m$ is reachable from $n$ then
            Then remove $n$ from flaggedPossPicky
        end
    end
end
```

The function EXISTSUCCESSOR is outlined in Algorithm 3. The cost of Algorithm 3 is $O(OUs)$, where O is the size of the output set, U is the size of the Unpicked set, and the determination of the successor relationship between u and o (line 5) takes times$O(s)$.

Finding the Frontier Picky Manipulation runs in $O(n{*}O{*}U{*}s+e)$ time where n is the number of manipulations in the DAG, e the number of edges in it, O is the largest output set for any manipulation in the workflow, and U and s are as above. The determination of Unpicked successors involves generating the relevant output set O_m, and then running the function successorExists(O_m, U_m) to determine if O_m includes any successor of the Unpicked U. (The algorithm for finding Frontier Picky manipulation also has a final step to ensure that there are no unpicked successors downstream of a possible Frontier Picky manipulation. Since the number of candidates is usually very small (and frequently is just one), we ignore this cost in the above formulae).

3.1.2 Top Down

An alternative strategy is to work top down from the result, looking for Unpicked successors. As soon as an Unpicked successor is sighted, back up one step and output the identified Frontier Picky manipulation. This top down strategy to find the Frontier Picky Manipulation, and thus the answer to WHY NOT?, is presented in Algorithm 2. It begins with the outputs of the penultimate manipulation and checks the lineage for every data item. If successors to the Unpicked are found, then the ultimate manipulation is the Fron-

[2] In the event that operator **1** also excluded (Euripides, “Medea”) from the intermediate result set, then the set of Frontier Picky Manipulations would be **1** and **2**.

tier Picky Manipulation. If no successors are found, the algorithm iteratively checks manipulations in reverse toplogically sorted order. This algorithm also runs in $O(n * s + e)$ time where n is the number of manipulations in the DAG, e the number of edges in it, and s is the time it takes to determine Unpicked successors.

Notice that this strategy requires all intermediate results to be stored after the query has completed for WHY NOT? query evaluation. While this is an unreasonable assumption in classic databases, it is often the normal operating procedure in e-science, non-database systems [6, 11, 21, 23].

Algorithm 2: Answering WHY NOT? Top Down.

```
Input: DAG, $M$, of manipulations, $m$
Input: Output Dataset, $O_m$, for each manipulation $m$
Input: Queue, $Q$, initialized with penultimate manipulations to Sink
Input: Unpicked, $U$
Output: Frontier Picky Manipulation(s), picky
# Run in Breadth First Search order from Sink to Source
forall $m$ manipulations $\in$ Queue $Q$ do
    $U_m$ = Unpicked inputs of $m$;
    if successorExists($O_m$, $U_m$) then
        forall $c$ manipulations $\in$ $m$.parents do
            $c$.numchild- -;
            if $c$.numchild == 0 then
                $Q$.add($c$);
            end
        end
    end
    else
        flagPossPicky($m$);
    end
end
forall $m$ manipulations flaggedPossPicky do
    forall $n$ manipulations flaggedPossPicky do
        if $m$ is reachable from $n$ then
            Then remove $n$ from flaggedPossPicky
        end
    end
end
```

3.1.3 Top Down vs. Bottom Up and Intermediate Datasets

The asymptotic worst case complexity of the two algorithms is identical. Which should one choose? It depends on where in the workflow the frontier is, and whether materialized intermediates exist. Top Down will quickly find Frontier Picky Manipulations close to the output, while Bottom Up will do better with earlier Frontier Picky Manipulations. In Bottom Up, we are faced with a distinct choice:

- Keep all intermediate result sets. Find the data items(s) in input and intermediate datasets that could correspond to it.
- Start with initial data items, and re-run, flagging all intermediates that are potential Unpicked data item(s).

There is obviously a trade-off in space and time for these two approaches. This has been explored in [7] in the form of Strong and Input-Only Identity, in which intermediate result sets are stored and only the input datasets are saved respectively. If all intermediates are stored, then we merely search through all input and intermediate data items for possible Unpicked matches using either Bottom Up or Top Down. On the other hand, if only input data items are kept, then we are required to utilize the Bottom Up approach, rerunning the set of MANIPULATIONs to obtain the required O_m for each. The Bottom Up algorithm presented above, assumes only input data is kept – if intermediate data is available, line 3 can be changed to avoid computing each O_m.

Algorithm 3: Code for successorExists function.

```
Input: $O$, the output set of some manipulation $m$
Input: Unpicked, $U_m$
forall $o$ data items $\in$ Dataset $O$ do
    forall $u$ data items $\in$ Unpicked $U_m$ do
        if $u \overset{m}{\angle} o$ then
            return TRUE;
        end
    end
end
return FALSE;
```

4. FINDING SUCCESSORS

The central function, repeatedly performed in the above algorithms, is the task of successor determination. For each item in the output set of each manipulation, in turn, we have to determine if it is the successor of some Unpicked item. The basic method of finding successor is to actually apply the manipulation with all inputs and compute the result. Given the result of interest, there is no easy way in general to go back and determine which source data items contributed to it, unless lineage information was being stored somewhere. Instead of laboriously checking lineage on every data item output from every manipulation, are there properties of manipulations that we can utilize to skip manipulations, or look at only a subset of outputs? What we want is Successor Visibility.

Given an input dataset, I, and output dataset, O, and a manipulation, m, for every data item produced by m, we can write $o_1 = m(i_1, i_2..., i_n)$ where $o_x \in O$ and $i_x \in I$.

DEFINITION 6. $Successor\ Visibility$:
A manipulation has Successor Visibility with respect to i_x if we can determine (for all values of i_x and o_y) whether there exist o_1, o_2, etc. such that $i_x \overset{m}{\angle} o_y$ $\forall i_x$ in $O(1)$ time.

In other words, if we can determine the successor of a data item after a manipulation, without performing the computation of the manipulation, or exploring alternative values for i_1, i_2, etc, then there is Successor Visibility. For relational operators like projection, the act of determining successor visibility will be with respect to a specific attribute. However, the above definition will also work for user defined functions (UDF). Consider a simple UDF that returns 1 if a word begins with ‘a’, 2 for ‘b’, etc. In this case, providing a hashmap can find a successor in O(1) time without specifying an attribute.

The definition of Successor Visibility merely states a desirable property, but does not specify how that property could be achieved. In a database query scenario, one common way to achieve this is through *attribute preservation*. For example, if our WHY NOT? question is with regard to the author Homer, and the author field is retained through the workflow from source dataset(s) to the result, then we say the attribute of interest has been *preserved*; it becomes trivial to identify Unpicked successors – they are exactly the intermediate (and final) result items with a value of “Homer” for the author attribute, no matter how may selections, projects, joins, etc. may have been applied.

Attribute preservation is not the only way to achieve Successor Visibility. In fact, many non-database workflows do not have this property. Consider a simple MANIPULATION found in myExperiment [23]: getProteinSequence. The input is a protein_id and the output is an amino acid sequence. In the above example, building a lookup hashtable of used protein_ids and sequences can permit successor determination in $O(1)$ time, and hence give Successor Visibility.

Manipulation	Visible?	Successor, o_x, given i
Projection	Yes	All o_x where the attribute-value set intersects the attribute-value set of i
Selection	Yes	All o_x with exact attribute-value matches to i
Rename	No	
Join	Yes	All o_x with intersecting attribute-value set on the "left" or"right" to i
Division	No	
MIN or MAX	Yes	o if o contains the attribute-value from i
COUNT, SUM AVERAGE	Yes	o

Table 2: The Visibility Rules for the Relational Operators. Given an input data item, in some cases, we can find successors without using the Lineage Method.

Moreover, a sequence of manipulations can have Successor Visibility if each manipulation in the sequence has Successor Visibility with respect to the appropriate (chain-forming) input. For instance, in the workflow in Figure 1(a), the module Select_All_Books takes in a data item from the books table, and produces an exact representation of it as a string. As such, it is possible to correlate the input and output data items without re-running the manipulation. Thus Select_All_Books has Successor Visibility. Indeed, the chain of manipulations from Select_All_Books through Select_Books_≤$20 has Successor Visibility. Notice that the manipulation Apply_SeasonalCriteria does not have Successor Visibility. In Table 2, we state conditions under which visibility can be used instead of lineage for standard relational operators.

Successor Visibility is not just a cute trick to make finding Successors faster. In some cases, it is the only way to find Successors. For user defined functions (UDF) in SQL, and any module in a workflow system such as [6, 11, 21], it is impossible to use lineage to trace data items through operators as we have in this work. Consider two workflow modules found in [23].

- A manipulation takes in a protein identifier, searches SwissProt and returns the protein record.
- A manipulation takes in the results from a NCBI query and removes duplicates.

Lineage as defined in [5, 9, 10] cannot be applied to these manipulations. However, both have Successor Visibility. Thus, by having Successor Visibility, our methods can be extended out of the white-box relational world into the black-box world of workflow systems. Of course, the obvious question that arrises is whether the majority of black-box manipulations in workflow systems have Successor Visibility. Our findings are positive. We sampled 100 workflows at random from myExperiments [23]. We found a total of 478 manipulations (called "modules" in their terminology) in these 100 workflows. Of the 478 modules, 273 easily satisfied Visibility in that they could be mapped to a relational algebra expression that has Successor Visibility. We believe that additional analysis can show that several of the remaining 205 modules also have Successor Visibility, but we did not perform an exhaustive analysis. Our point is simply that a majority of the modules were Successor Visibile. Moreover, work such as [25] is attempting to make workflow modules more visible by tracing and understanding the underlying operating system calls. Thus, Successor Visibility can be used in many cases to answer WHY NOT? questions in workflow systems.

Title	Author	Price	Main Char.	Pub. Date	Loc.
Anna Karenina	?	?	Woman	1800s	Rus.
?	J.R.R Tolkien	$30	Wizard	1940s	UK
Hamlet	Shakespeare	?	King	1600s	UK
Harry Potter 1-7	J.K Rowling	?	Wizard	1990s	UK
Antigone	Sophocles	?	Woman	400BC	?
Aeneid	Virgil	?	Warrior	100BC	?
Hitchhikers Guide to the Universe	?	$5.99	Aliens	1970s	?
Odyssey	Homer	$49	King	800BC	?
Basilius	Hrotsvit	$20	Woman	900s	?
Pride and Prejudice	Jane Austin	$14.99	Woman	1800s	UK

Table 3: The Knowledge Table presented to the users. "?" indicate values the user does not know.

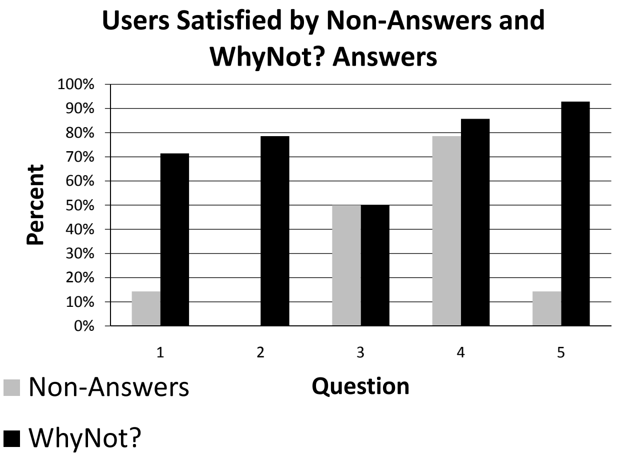


Figure 2: The users who were satisfied with Non-answers and WHY NOT? answers for each query in Table 4.

5. EVALUATION

Is it possible to provide useful WHY NOT? answers to users? Can we compute these answers in reasonable time? Our findings are positive.

In this section, we demonstrate the feasibility of WHY NOT? answers. We begin with a user evaluation of WHY NOT? answers, and show that users both find the style of answers presented in this work useful and informative. We then evaluate the efficiency of alternative techniques for finding WHY NOT? answers.

5.1 User Evaluation

5.1.1 Setup

This work proposes a methodology for answering user questions about their world. As such we designed a user study to evaluate the effectiveness of WHY NOT? answers. While we have anecdotal support from scientists that this approach is useful, it was impossible to build a controlled experiment separated from individual scientist's domains of interest. As such, we used the running book example as a domain independent base for control reasons.

	English Query	Behind the Curtain Execution	Eval. Plan
1	Select all "Window-Books" (you think window books are ≤ $20)	bookstore ⟶ (select books ≤ $20) ⟶ (select books written after 1700) ⟶ output results	Fig. 1(a)
2	Select the top count character-type	bookstore ⟶ (select book characters) ⟶ (group kings/warriors/wizards as king) ⟶ (for characters, tally count) ⟶ (select top count) ⟶ output results	Fig. 1(b)
3	Select all books whose title starts with 'A' and are priced less than the Odyssey	bookstore ⟶ [(select books title 'A' = a), (select price of the Odyssey = b)] ⟶ (retain b > a.price) ⟶ output results	Fig. 1(d)
4	Select all books written in the UK and price < $25	bookstore ⟶ [(select books < $25 as a), (select books from UK as b)] ⟶ (join a,b) ⟶ output results	Fig. 1(e)
5	Select the union of Shakespeare books and books that cost < $50, that are both a window book and published before 1950.	[bookstore ⟶ (select books < $50 as a), bookstore ⟶ (select books author='Shakespeare' as b)] ⟶ (union a,b) ⟶ (select pub before 1950) ⟶ (select college-book) ⟶ (select books ≤ $20) ⟶ (select books written after 1700) ⟶ output results	Fig. 1(c)

Table 4: The set of English language queries users were asked to perform, and the secret, behind the curtain execution that actually occurred.

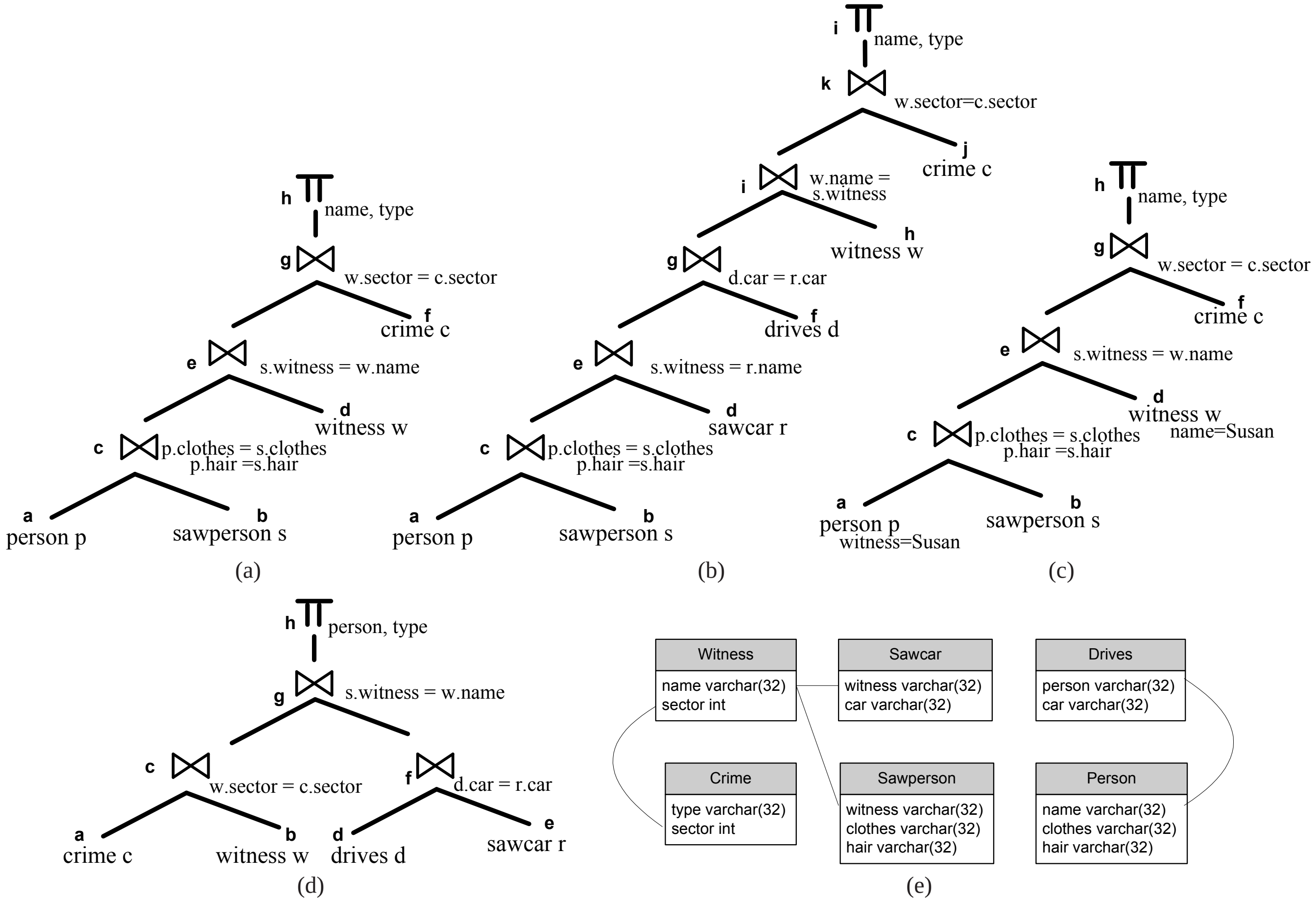


Figure 3: (a)-(d) The query evaluation plans for the Crime Queries used (Queries 1,3,4,2 respectively). (e) The Trio Crime Schema.

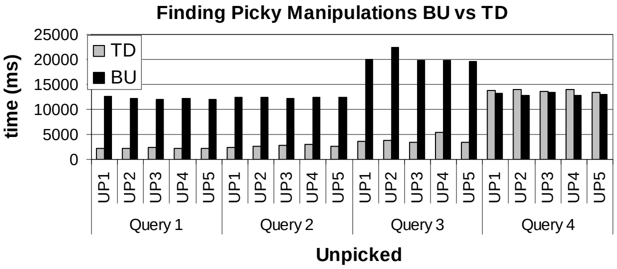


Figure 4: Finding the Frontier Picky Manipulation for an Unpicked Set using the BU or TD algorithm using Lineage.

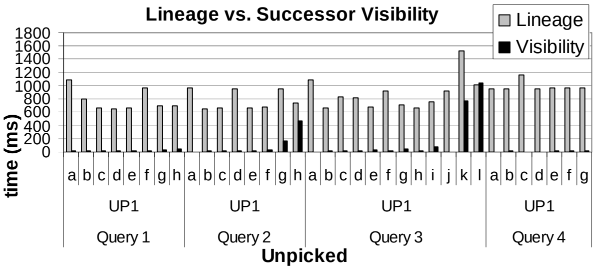
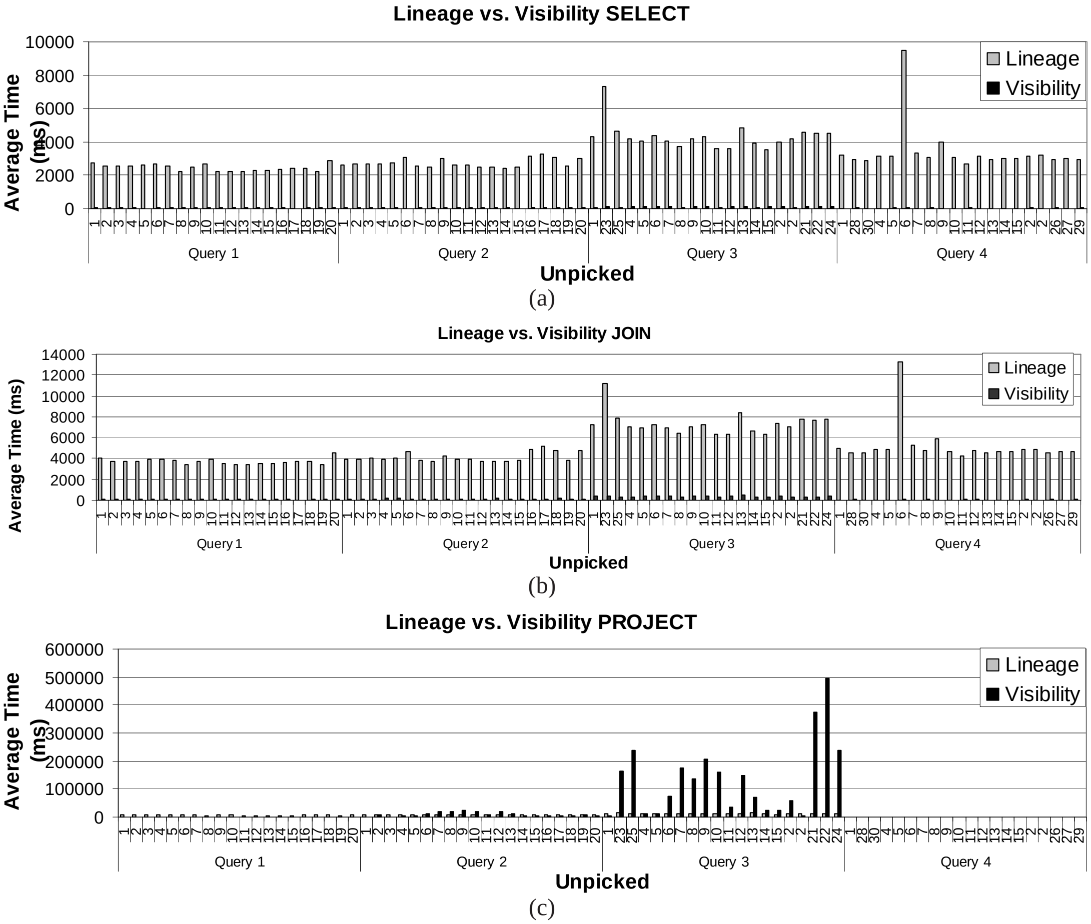


Figure 5: Using Lineage vs. Successor Visibility to find the existence of the Unpicked in the manipulation's output.



(a)

(b)

(c)

Figure 6: Lineage and Visibility time for each operator within Query 1 for a single and multiple Unpicked set.

Because we are interested in user questions about what is *Not* in the result set and this is dependant on expectations of query results, the first hurdle was to force a disparate group of users to ask the same set of WHY NOT? questions. Users were presented with a Knowledge Table, as shown in Table 3, that represented their knowledge of the Book Database. They were asked not to rely upon personal knowledge of any of the books. The users were then given a set of English language queries to find books in the Book Database. Table 4 shows the queries the users were given. Users were told that the English query expressed was what they wished to find books for, but that behind the curtain, the executed query could do extra machinations. The exact evaluation plan or workflow is described in Table 4 as well, and except for minor manipulation differences[3] these mimic the workflows and evaluation plans in Figure 1.

To force the users to think about the query, they were then asked to mark the checkbox of each tuple that *should* be in the result set based on the English language query. In the worst case, 87% of the users chose the expected result set that satisfied the query. In the best case, 100% did. Thus, the bulk of the users expected the same set of tuples in the result set. The users were then presented with the actual result set of the query. These were designed to contain all but one of the tuples from the expected result set, thus encouraging all disparate users to ask WHY NOT? about a particular tuple. When users asked WHY NOT? about this tuple, they were presented with the answer given by our technique and by a comparison baseline technique.

There is very little work addressing the WHY NOT? problem. The only solid comparable work we are aware of is in [14]. In that work, which was focused on explaining why information is not present after querying extracted and integrated data from the web, the authors look at non-answer tuples in the input sets that could become answers (but do not). They define trusted and untrusted sources, then find if there are any valid updates to the untrusted sources that will include the sought after tuple in the result set. In other words, looking at the non-answer (Virgil, "Aeneid") after the query plan in Figure 1(d), updating the price value to $<$ \$49 would include (Virgil, "Aeneid") in the result set. Thus, the "answer" presented in [14] would be the set of changes to the non-answer tuple that would have resulted in its inclusion in the result set; we will call this method *Non-answers*. The second answer presented was that of our technique, the Frontier Picky Manipulation; we refer to this method as WHY NOT?. For each answer, the user was asked whether the answer satisfied their question about the missing tuple.

Fourteen users partook in the user evaluation. All were experts in Computer Science, most with graduate degrees, but none knew the details of the problem we were addressing, or were aware which technique was ours.

5.1.2 Satisfaction

Figure 2 shows the user satisfaction for each query. In total, 76% of users were satisfied on average with WHY NOT? answers. While, this is not quite 100%, it is a large majority, and is probably good enough for us to declare that our proposed technique is useful. In comparison, 31% of users were satisfied with Non-answers as explanation. Further analysis reveals other patterns.

Given the set of user Questions, and the actual execution plans presented in Table 4, two classes of questions are evident. Queries 3–4 have relatively straightforward English language Queries and simple evaluation plans. On the other hand, Queries 1, 2 and 5 are more complex. In Queries 1–2, the English Query is straightforward, but the User Defined Functions actually executed do not align exactly with the English Query. Thus, the users are at a loss when something so simplistic looking doesn't behave as expected. Query 5 is complicated in a different way; the English Query matches the evaluation plan exactly, but is very complex and hard to follow. In Figure 2, there are two distinct behaviours visible. For Questions 1,2 and 5, WHY NOT? does significantly better than non-answers, while for Questions 3–4, non-answers and WHY NOT? are on par. In other words, while non-answers provides adequate answers for straightforward selection-style queries, WHY NOT? is significantly superior (with a p-value of $2.3e - 7$ according to a Fisher Exact Test) when complications in either the query or the execution plan are present.

5.2 Performance

As discussed in Sections 3–4, there are two methods for finding Frontier Picky Manipulation(s), Top Down and Bottom Up, and two methods for finding successors: lineage and visibility. To date, there is only one system we are aware of that supports lineage as a first class operator, Trio. Trio [1, 19] is built on top of Postgres, and has the ability to trace the lineage of any tuple found in a view back to the original input tuple(s). Since one of the methods proposed for finding successors requires lineage, we used Trio as our backend database. All algorithms were implemented in Java and run on a Dell Windows XP workstation with Celeron(R) CPU at 3.06GHz with 1.96GB RAM and 122GB disk space. For comparison of Bottom Up vs. Top Down, materialized intermediates were utilized.

We utilized the Crime queries that are so often used to showcase Trio, since they had complex query evaluation plans that could provide a variety of answers for WHY NOT? questions. Additionally, while we used the classic Crime dataset as a template, we expanded the number of tuples so that it was less of a toy dataset. The total size of the crime database is 4MB. Queries 1–4 produce 7695, 65,319, 140,699 and 5 tuples respectively. The numbers presented below rely heavily upon the specifics of the Trio system. However, Trio is the only system available for tracking lineage within a database.

We ran four base queries, performed against the expanded Trio crime dataset. The evaluation plans for all four queries were determined using "Explain", and are shown in Figure 3. For each query, we then asked a series of WHY NOT? questions by specifying an attribute that existed in the input dataset but not in the final result set. For instance, WHY NOT? "Mary", where Mary could be a potential value for a suspect or witness.

5.2.1 Bottom up vs. Top Down

Figure 4 shows the run times to find the Frontier Picky Manipulation given an Unpicked set using either the Bottom Up (BU) or Top Down (TD) approach, using lineage to find Unpicked Successors.

TD does significantly better than BU for all query evaluation plans except Query 4. Given the nature of the query evaluation plans, this is to be expected. Consider the query evaluation plan for Query 1 in Figure 3(a), and the Unpicked data item UP1, "Antigone". There are only five tuples in the entire crime database that can be mapped to an Unpicked with "Antigone": a tuple from the Witness table with Witness.name="Antigone", three tuples from the Sawcar table with Sawcar.witness="Antigone" and one tuple from the Sawperson table with Sawperson.witness="Antigone" (schema for the crime database is in Figure 3(e)). UP1 does not ever exist in

[3] Minor changes were made to the plans in Figure 1 for representation ease and to decrease the amount of reading required of the users. For instance, Window Books are books <\$20 and published after 1700, instead of <\$20 and fitting a seasonal criteria based on a particular date.

manipulations **a**, or **f** in Figure 3(a). However, these are not Frontier Picky Manipulations since it does exist in another set of paths, **b**, **c**, **d**, **e**, and **g**. The true Frontier Picky Manipulation is **h** since this is where the attribute "Antigone" finally disappears from the result set. As such, the TD algorithm only tests one manipulation, while the BU algorithm must work through all eight. Conversely, Query 4 is highly selective very near the sources. As such, TD must check all eight manipulations. BU does not save very much though. Although the Frontier Picky manipulations are very close to the sources, BU still must check four of the manipulations. Thus using the TD algorithm is best when there is low selectivity early in the query evaluation plan. However, while TD may be worse than BU when there is high selectivity early in the evaluation plan, it is not much worse since BU must do almost as much work. In other words, due to the "fan" structure of most query plans, TD should be preferred to BU, in situations where both can be used.

5.2.2 *Lineage vs. Successor Visibility*

The times presented thus far are using lineage to determine Unpicked successors. If we have Successor Visibility, using attribute preservation for the Crime queries, then we can find successors much more efficiently, as discussed in Section 4. Figure 5 shows these savings. The labels on the X-axis map to the manipulations labeled for each Query in Figure 3. Overall, using Successor Visibility causes a marked decrease in the amount of time needed to detect Unpicked Successors, as expected. However, this turns out not always to be a win – keeping track of Successor Visibility requires additional data structure support. For example, in Query 3, manipulation **l** is lineage equal to Successor Visibility. This manipulation (and **k** before it) is dealing with 140,699 tuples, and the data structures used to implement Successor Visibility add overhead while providing only limited benefit.

In Figure 6 we show the average time for lineage and visibility for all queries and Unpicked data items run in all experiments broken down by relational operator type. For all selection and join operators, using Successor Visibility does much better than lineage. However, for projections in Query 3, using lineage is better. Remember that Query 3 generates a huge result set, and the structures used for Successor Visibility begin to thrash at that level of output. Eventhough Successor Visibility is a fast operation, it is performed outside the relational database, thus losing all performance enhancements for large data manipulation that can be leveraged by the lineage version. Successor Visibility in this case could perform on par with lineage if more attention were paid to memory management, disk io, etc. Otherwise, using Successor Visibility enables a drastic reduction in time needed to find the Frontier Picky Manipulation.

5.2.3 *Size of the Unpicked Set*

In Figures 4–5, each Why Not? question identified a small set of tuples in the input data set as Unpicked, on average about 5 items. Figure 7 shows how the Why Not? algorithms fare with a change in the number of Unpicked data items. For clarity we show only the results from Query 4, and compare against Why Not? questions that have only a few Unpicked Data items. Unpicked UP1–5 have five Unpicked tuples returned, while Unpicked UP26–30 specify the attributes most found in the database, returning up to 50 Unpicked tuples. We find that the number of Unpicked tuples does not affect the overall runtime of either TD or BU algorithms, whether we use Lineage or Successor Visibility.

These experiments show that it is possible to answer Why Not? queries in reasonable time on basic desktop machines and the cost of our algorithms is generally small compared to the cost of the obtaining lineage. If we were to use a different system to manage lineage, the performance would largely depend on this underlying system, and not on our algorithms.

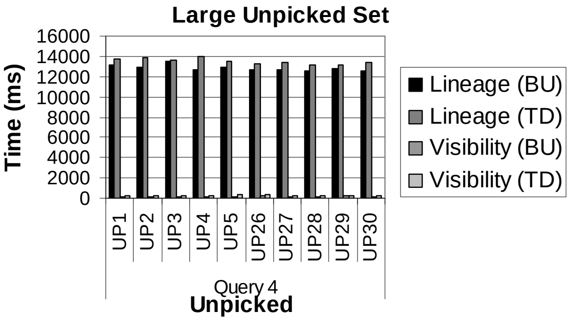


Figure 7: Effect of a Large Unpicked set on finding the Why Not? answer.

6. RELATED WORK

This work draws heavily upon the formalisms and concepts of lineage set out by [8, 9, 10], and uses the implementation of them in Trio [1, 19, 24]. Moreover, work such as [25], is attempting to extend the ability of tracing lineage through non-relational operators by recording system-level calls and recording what happens for each input despite not being able to see in the workflow-module black box.

Several groups are also beginning to think about why items are not in the result set. For instance, [14] defines the concept of the provenance of non-answers. A non-answer, is very similar to our concept of an Unpicked Data item. However, instead of attempting to find the manipulation that excluded it from the result set, [14] look for the *attribute* within the Unpicked that caused it to be excluded from the result set. By substituting an "always true" value for each attribute in the tuple until it is included in the result set, they can pinpoint the attribute(s) responsible.

This work attempts to answer user queries about results that are created via processes and datasets that are opaque to the user. [16, 20] attempt to do this for programmatic interference. For example, "Why did MSWord capitalize this word?". While the details of how they accomplish this task are completely different from ours, the underlying problem remains the same: users are confronted daily with processes and data that they do not understand. Additionally, [22] looks at data publishing security, and allows users to verify that their query results are complete as well as authentic. While their motivation and methods are security focused, they too are attempting to give users more control and ability to probe the underlying data.

7. CONCLUSIONS

In this work, we outline a new problem facing users whose access to the underlying data is restricted. When users are unable to sift through the data themselves, it is impossible to discover why a data item is not in the result set. Is it not in the input datasets? Is some manipulation between the input and the user discarding it? Etc. In particular, we found that when biological research scientists were presented with information from bioinformatics tools that clashed with their beliefs, especially absent data, they were unable to determine what happened. We provide a framework that allows users to ask WHY NOT? questions about data items not in the result set (Unpicked). Additionally, we create a model that allows us to pinpoint where the Unpicked data item was discarded from the final result set.

We implement the model using two different algorithms for finding the manipulation of interest, and two different methods for finding a data items's successor. We show how these methods compare using a well-known set of queries. Additionally, utilizing a user study, we show that the techniques presented herein satisfactorally answer a user's WHY NOT? questions.

Our goal in this work was to create a practically useable user understanding system for databases of moderate size. Explanations are expensive, and scaling this approach to large databases is a challenge, and will probably involve special consideration of particular operators instead of generic manipulations as done in this work. In this work, we have found a sweet spot in which we can give meaningful explanations, at acceptable time and computational cost.

8. ACKNOWLEDGMENTS

This work was supported in part by NSF grant number IIS 0741620 and by NIH grant number U54 DA021519.

9. REFERENCES

[1] Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB Seoul, Korea*, pages 953–964, 2006.

[2] Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. In *VLDB*, pages 900–911, 2004.

[3] Shawn Bowers, Timothy McPhillips, Martin Wu, and Bertram LudÃd'scher. Project histories: Managing data provenance across collection-oriented scientific workflow runs. In *DILS*, pages 27–29, 2007.

[4] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang-Chiew Tan. Archiving scientific data. In *SIGMOD*, pages 1–12, June 2002.

[5] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[6] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, and Cláudio T. Silvaand Huy T. Vo. VisTrails: Visualization meets data management. In *SIGMOD*, pages 745–747, 2006.

[7] Adriane Chapman and H.V. Jagadish. *Provenance and the Price of Identity*, pages 162–170. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Provenance and Annotation of Data edition, 2008.

[8] Y. Cui, J. Widom, and J.L. Wiener. Tracing the lineage of view data in a data warehousing environment. In *ACM Transaction on Database Systems (TODS)*, 2000.

[9] Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In *ICDE*, pages 367–378, 2000.

[10] Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. In *VLDB*, pages 41–58, 2001.

[11] Susan Davidson, Sarah Cohen-Boulakia, Anat Eyal, Bertram Ludascher, Timothy McPhillips, Shawn Bowers, and Juliana Freire. Provenance in scientific workflow systems. *IEEE Data Engineering Bulletin*, 32(4):44–50, 2007.

[12] Ian Foster, Jens Vockler, Michael Eilde, and Yong Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *SSDBM*, pages 37–46, July 2002.

[13] Paul Groth, Simon Miles, Weijan Fang, Sylvia C. Wong, Klaus-Peter Zauner, and Luc Moreau. Recording and using provenance in a protein compressibility experiment. In *HPDC*, 2005.

[14] J. Huang, T. Chen, A. Doan, and J. Naughton. On provenance of non-answers to queries over extracted data. In *VLDB*, 2008.

[15] Magesh Jayapandian, Adriane Chapman, et al. Michigan Molecular Interactions (MiMI): Putting the jigsaw puzzle together. *Nucleic Acids Research*, pages D566–D571, Jan 2007.

[16] Andrew J. Ko and Brad A. Myers. Designing the whyline: a debugging interface for asking questions about program behavior. In *SIGCHI*, pages 151–158, 2004.

[17] Simon Miles, Sylvia C. Wong, Weijian Fang, Paul Groth, Klaus-Peter Zauner, and Luc Moreau. Provenance-based validation of e-science experiments. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(1):28–38, 2007.

[18] Luc Moreau, Bertram Ludäscher, et al. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 2007. http://twiki.ipaw.info/bin/view/Challenge/SecondProvenanceChallenge .

[19] Michi Mutsuzaki, Martin Theobald, et al. Trio-One: Layering uncertainty and lineage on a conventional DBMS. In *CIDR*, pages 269–274, 2007.

[20] Brad A. Myers, David A. Weitzman, Andrew J. Ko, and Duen H. Chau. Answering why and why not questions in user interfaces. In *SIGCHI*, pages 397–406, 2006.

[21] Tom Oinn, Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, et al. Taverna: lessons in creating a workflow environment for the life sciences: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1067–1100, 2006.

[22] H. Pang, A. Jain, K. Ramamritham, and K. Tan. Verifying completeness of relational query results in data publishing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2005.

[23] D. De Roure and C. Goble. myExperiment - a web 2.0 virtual research environment. In *International Workshop on Virtual Research Environments and Collaborative Work Environments*, 2007.

[24] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.

[25] Mingwu Zhang, Xiangyu Zhang, Xiang Zhang, and Sunil Prabhakar. Tracing lineage beyond relational operators. In *VLDB*, pages 1116–1127, 2007.