Data X-Ray: A Diagnostic Tool for Data Errors

Xiaolan Wang
University of Massachusetts
xlwang@cs.umass.edu

Xin Luna Dong
Google Inc.
lunadong@google.com

Alexandra Meliou
University of Massachusetts
ameli@cs.umass.edu

ABSTRACT

A lot of systems and applications are data-driven, and the correctness of their operation relies heavily on the correctness of their data. While existing data cleaning techniques can be quite effective at purging datasets of errors, they disregard the fact that a lot of errors are *systematic*, inherent to the process that produces the data, and thus will keep occurring unless the problem is corrected at its source. In contrast to traditional data cleaning, in this paper we focus on *data diagnosis*: explaining where and how the errors happen in a data generative process.

We develop a large-scale diagnostic framework called DATAXRAY. Our contributions are three-fold. First, we transform the diagnosis problem to the problem of finding common properties among erroneous elements, with minimal domain-specific assumptions. Second, we use Bayesian analysis to derive a cost model that implements three intuitive principles of good diagnoses. Third, we design an efficient, highly-parallelizable algorithm for performing data diagnosis on large-scale data. We evaluate our cost model and algorithm using both real-world and synthetic data, and show that our diagnostic framework produces better diagnoses and is orders of magnitude more efficient than existing techniques.

1. INTRODUCTION

Systems and applications rely heavily on data, which makes data quality a detrimental factor for their function. Data management research has long recognized the importance of data quality, and has developed an extensive arsenal of data cleaning approaches based on rules, statistics, analyses, and interactive tools [1, 25, 35, 50, 51]. While existing data cleaning techniques can be quite effective at purging datasets of errors, they disregard the fact that a lot of errors are *systematic*, inherent to the process that produces the data, and thus will keep occurring unless the problems are corrected at their source. For example, a faulty sensor will keep producing wrong measurements, a program bug will keep generating re-occurring mistakes in simulation results, and a bad extraction pattern will continue to derive incorrect relations from web documents.

In this paper, we propose a data diagnostic tool that helps data producers identify the possible systematic causes of errors in the data. This is in contrast to traditional data cleaning methods, which treat the symptom (the errors in a dataset) rather than the underlying condition (the cause of the errors). Since finding particular causes is often domain specific, we instead aim to provide a generic approach that finds groupings of errors that may be due to the same cause; such groupings give clues for discerning the underlying problems. We call our tool DATAXRAY: just as a medical X-ray can facilitate (but not in itself give) the diagnosis of medical conditions, our tool shows the inherent relationship between errors and helps diagnose their causes. We use examples from three different domains to illustrate how DATAXRAY achieves this goal.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2750549.

EXAMPLE 1 (KNOWLEDGE EXTRACTION). *Web-scale knowledge bases [20,24,34] often contain a large number of errors, resulting both from mistakes, omissions, or oversights in the knowledge extraction process, and from erroneous, out-of-date information from the web sources. Existing knowledge curation techniques focus on finding correct knowledge in the form of (subject, predicate, object) triples by analyzing extractions from multiple knowledge extractors [20, 21]; however, they do not offer any insight on why such errors occur and how they can be prevented in the future.*

We applied DATAXRAY *on 2 billion knowledge triples from Knowledge Vault [20], a web-scale probabilistic knowledge base that continuously augments its content through extraction of web information, such as text, tables, page structure, and human annotations. Our tool returns groupings of erroneous triples that may be caused by the same systematic error. Among the many different types of errors that* DATAXRAY *reports, we give three examples.*

Annotation errors: DATAXRAY *reports a grouping of about 600 knowledge triples from besoccer.com with object "Feb 18, 1986", which were extracted from webmaster annotations according to schema.org; among them the error rate is 100% (all triples conflict with the real world). A manual examination of three to five instances quickly revealed that the webmaster used* "2/18/1986" *to annotate the date of birth for all soccer players, possibly by copying HTML segments.*

Reconciliation errors: DATAXRAY *reports a grouping of about 700,000 triples, extracted by a particular extractor from various websites, with object "baseball coach"; among them the error rate is 90%. Manual investigation showed that the mistakes resulted from reconciliation errors: all* coaches *were reconciled to* baseball coaches.

Extraction errors: DATAXRAY *reports a grouping of about 2 million triples, extracted by a particular extractor from various websites, with predicates containing "olympics"; among them the error rate is 95%. Manual investigation showed that the mistakes resulted from an over-generalized extraction pattern that extracts all sports games as olympic games.*

These three examples illustrate that the errors exposed by our tool cover a variety of problems, from data errors to extraction errors. Some of these problems can be very hard to detect; for example, web annotations are typically not visible from webpages. Even though DATAXRAY *does not report the causes directly, it guides diagnosis by reporting sets of errors that are likely due to the same cause.*

EXAMPLE 2 (WIRELESS PACKET LOSS). *A wireless sensor network experiences significant packet loss.* DATAXRAY *reports a grouping of messages containing the range of node IDs 10–15 as destinations, where the message drop rate is very high. Manual investigation unveils that nodes 10–15 are all located on a side of the building with poor connectivity due to interference.*

EXAMPLE 3 (TRAFFIC INCIDENTS). *We use* DATAXRAY *to analyze traffic incident and weather data collected by the US department of Transportation on multiple freeways in Portland, OR, over two months [?]. Our algorithm uses the reported traffic incidents as "error" labels, and automatically derives that surface water level of more than 2cm is likely a cause of accidents.*

Diagnosing errors in big data environments raises three major challenges that make existing techniques, such as provenance analysis, feature selection, and causal analysis, not applicable.

Massive scale. Many applications, such as knowledge extraction and sensing, continuously produce huge volumes of data. A sampling that results in a manageable data size often loses statistical strength. Algorithms working with data of this size require linear time complexity and the ability to process data in parallel. Existing techniques such as feature selection [46, 54] cannot handle this increased scale, because they are not easy to implement in shared-nothing architectures [40].

System complexity. Data generative processes, such as knowledge extractors, often implement a large number of algorithms, invoke external tools, set different parameters, and so on. As a result, it is not feasible to analyze them and reason with them directly. Existing provenance techniques [15, 19] are not well-suited for this level of complexity, and most existing tools on data auditing focus on much simpler, relational settings [30, 52].

High error rates. In some applications such as knowledge extraction from the web, the error rate can be as high as 70% [21]. This disqualifies causal analysis techniques [42, 43], whose premise relies on the assumption that errors, and generally observations that require explanation, are rare.

DATAXRAY addresses these challenges by analyzing the relationship between erroneous instances in an efficient and scalable manner. More concretely, we make the following contributions.

- We abstract the processes that derive data using a hierarchical structure of *features*. Each feature corresponds to a subset of data properties and each cause of errors can be considered to be associated with a feature. We then transform the problem of error diagnosis to the problem of finding the features that best represent erroneous elements. This transformation enforces minimal assumptions, can model a large range of application scenarios, and allows for efficient exploration of possible diagnoses (Section 2).
- We apply Bayesian analysis to estimate the *causal likelihood* of a set of features being associated with the causes of the errors, and use that to determine the most likely diagnosis for a given set of errors. We identify three intuitive principles for good diagnoses: conciseness (simpler diagnoses are preferable), specificity (each diagnosis should be closely associated with the real cause), and consistency (diagnoses should not be contradicted by a lot of correct data). We design a cost model that captures these principles and can be evaluated efficiently (Section 3).
- We exploit the hierarchical structure of features and propose a top-down iterative algorithm with linear time complexity that evaluates possible diagnoses from broader to more concise using our cost model. We then extend our algorithm to a parallel, MapReduce-based version (Section 4).
- Our evaluation includes three phases of experiments. First, we evaluate our cost model on real-world extraction data and demonstrate that it is significantly more effective at deriving diagnoses than other feature selection methods, including logistic regression. Second, we show that our algorithm is orders of magnitude more efficient than other feature selection methods. Third, we present experiments on synthetic data demonstrating that our approach can scale effectively to very large data sizes (Section 5).

2. DATA MODEL ABSTRACTIONS

In this section, we introduce a running example motivated by knowledge extraction, and describe a model abstraction to formalize the problem of diagnosing errors in this setting. Although we focus on knowledge extraction as the driving application for our framework, our approach can easily adapt to general settings.

EXAMPLE 4. *Figure 1a depicts two example web tables that reside on the same wiki page, containing information about musicians. Figure 1b depicts the knowledge triples extracted from these web tables using an extraction system. For example, the triple (P. Fontaine, DoB, c.1380) represents the information that the date of birth of P. Fontaine is c. 1380.*

Some of the extracted triples are incorrect, and are highlighted in the table of Figure 1b (t_5, t_9, and t_{12}). While traditional cleaning techniques may simply remove these triples from the knowledge base, or further provide a list of such triples as feedback, our objective is to help diagnose *the problem and understand the reasons for these errors. In this case, the reason for the incorrect results is that the extractors assign a default value ("01/01/1900") to unknown dates.*

In this work, we assume that we know which extracted triples are incorrect. Such labels can be obtained by existing cleaning and classification techniques [21, 25, 26]. They may also occur naturally in the data, such as extraction confidence from the extraction systems, or the occurrence of accidents in Example 3. *Our goal is not to identify the errors, but to reveal common properties of the errors that may be helpful in diagnosing the underlying causes.*

2.1 The element-feature model

We observe that a *cause* of errors is often associated with some *properties* of the erroneous instances and causes a high error rate for data with these properties. In Example 4, the three erroneous triples are caused by using *01/01/1900* as the default value when a date is unknown. Indeed, they share a common property that their objects are all *01/01/1900*. By highlighting the observation that the error rate is high (1.0 in our example) for triples with "object value: 01/01/1900", we can help users diagnose this possible cause. As another example, imagine a high error rate for triples extracted from *Tbl #1* (Figure 1b) where the objects are of the *Date* type. It suggests that the date format in that table may not be captured properly by the extractors. Surfacing such observation for triples with "source tableID: Tbl # 1" and "object type: date" can help the diagnosis.

Based on this intuition, we define the *element-feature model*, where we consider each data instance as an *element*, and capture its properties by a set of *property* values. We then use a subset of property values, which we call a *feature*, to capture a possible cause of errors. Features can be derived from data using their schema, types, and values, as well as from provenance metadata. Features form a

Musicians – Table 1

Name	Date of Birth	Date of Death
P. Fontaine	c.1380	c.1450
J. Vide	unknown	1433

Composers – Table 2

Name	Date of Birth	Date of Death
G. Legrant	fl.1405	N/A
H. Lantins	fl.c.1420	unknown

(a) Two web tables with information about musicians, that appear on the same wiki page.

ID	knowledge triple	*source* URL	*source* tableID	*subject* type	*subject* instance	*predicate* type	*predicate* instance	*object* type	*object* instance
t_1	{P. Fontaine, Profession, Musician}	wiki	tbl #1	People	P. Fontaine	Bio	Profession	Profession	Musician
t_2	{P. Fontaine, DoB, c.1380}	wiki	tbl #1	People	P. Fontaine	Bio	DoB	Date	c.1380
t_3	{P. Fontaine, DoD,c.1450}	wiki	tbl #1	People	P. Fontaine	Bio	DoD	Date	c.1450
t_4	{J. Vide, Profession, Musician}	wiki	tbl #1	People	J. Vide	Bio	Profession	Profession	Musician
t_5	{J. Vide, DoB, 01/01/1900}	wiki	tbl #1	People	J. Vide	Bio	DoB	Date	01/01/1900
t_6	{J. Vide, DoD, 1433}	wiki	tbl #1	People	J. Vide	Bio	DoD	Date	1433
t_7	{G. Legrant, Profession, Composer}	wiki	tbl #2	People	G. Legrant	Bio	Profession	Profession	Composer
t_8	{G. Legrant, DoB, fl.1405}	wiki	tbl #2	People	G. Legrant	Bio	DoB	Date	fl.1405
t_9	{G. Legrant, DoD, 01/01/1900}	wiki	tbl #2	People	G. Legrant	Bio	DoD	Date	01/01/1900
t_{10}	{H. Lantins, Profession, Composer}	wiki	tbl #2	People	H. Lantins	Bio	Profession	Profession	Composer
t_{11}	{H. Lantins, DoB, fl.c.1420}	wiki	tbl #2	People	H. Lantins	Bio	DoB	Date	fl.c.1420
t_{12}	{H. Lantins, DoD, 01/01/1900}	wiki	tbl #2	People	H. Lantins	Bio	DoD	Date	01/01/1900

(b) Knowledge triples extracted from the web tables in (a) and values of their properties in all dimensions. Incorrect triples are highlighted.

Figure 1: Error diagnosis in information extraction motivated by Example 1. An extraction pipeline processes the web tables in (a) and derives 12 knowledge triples (elements). Each element has four property dimensions with different granularity levels. The extractors assign a default value to dates that are unknown ("01/01/1900"), leading to three highlighted incorrect triples (t_5, t_9, and t_{12}).

natural hierarchy based on containment relationships; for example, the feature with object_type=Date contains, and thus is an ancestor of, the feature with object_type=Date $\wedge$ object_instance=01/01/1900. The problem of error diagnosis then reduces to the problem of finding a set of features that best summarizes all erroneous data elements. We next define the terms used in this model.

Property dimension: A property dimension describes one aspect of a data instance. For our example dataset, there can be four dimensions: source, subject, predicate, and object. Without losing generality, we consider a certain ordering of the dimensions.

Property hierarchy: Recall that features in each dimension form a hierarchy. In our example, the source dimension has three levels in the hierarchy: Root, source_URL, source_tableID. Each other dimension also has three levels; taking subject as an example, the hierarchy is Root, subject_type, subject_instance.

Accordingly, we define the property hierarchy as follows. The root of the hierarchy represents the coarsest granularity and each dimension has value ALL for the root level (Root). Descendant properties are finer-granularity representations of their ancestors; we say property A is a child of property B if one of the features of A is a child of the corresponding feature of B. For example, property {ALL, ALL, ALL, (object_instance, 01/01/1900)} is a child of property {ALL, ALL, ALL, (object_type, Date)}. As we show later, the hierarchy will allow us to solve the problem efficiently, using an algorithm that explores it in a top-down fashion (Section 4).

Property vector: With property dimensions and hierarchies, we can use a property vector to represent a data instance or a set of instances that share common properties. The vector contains one (property, value) pair for each dimension, where the property is in the hierarchy of that dimension, and the value is for the particular property. The root-level property corresponds to the pair (Root, ALL), but we write just ALL for short. For example:

- {ALL, ALL, ALL, (object_instance, 01/01/1900)} represents all triples with object 01/01/1900.
- {(source_tableID, Tbl#1), ALL, ALL, (object_type, Date)} represents all triples from *Tbl#1* of the particular *wiki* page with objects of the *Date* type.

Element: For each data instance (triple) we define an element to capture its truthfulness and property vector; the vector should contain a value for the leaf-level property for every dimension.

DEFINITION 5 (ELEMENT). *Consider a dataset with m property dimensions. A data instance is an element $e = (V, P)$, where*

- *V is `true` if the instance is correct and `false` otherwise;*
- *$P = \{d_1, \ldots, d_m\}$ is a property vector, where $d_i, i \in [1, m]$, is a (property, value) pair for the leaf property of the i-th dimension.*

Figure 2a presents a subset of the elements that correspond to the triples in Figure 1b.

Feature: Each property vector defines a set of triples that share a set of properties; we call it a *feature*.

DEFINITION 6 (FEATURE). *Consider a dataset with m property dimensions. A Feature f is a pair $f = (P, \mathbf{E})$, where*

- *$P = \{d_1, \ldots, d_m\}$ is a property vector, where $d_i, i \in [1, m]$, is a (property, value) pair for a property in the hierarchy of the i-th dimension.*
- *$\mathbf{E}$ is the set of elements with the properties represented by P.*

Figure 2b shows some example features for Example 4. As an example, feature f_6 represents all triples whose object is *01/01/1900*; elements e_5, e_9, and e_{12} carry this feature.

Problem definition.

We now formalize the problem of deriving diagnoses for data errors using the element-feature model. Each feature identifies a possible cause of error, and a diagnosis is a set of features that collectively explain the causes of the observed errors.

Element	V	Property Vector			
e_1	`true`	{(source_tableID, tbl #1),	(subj_instance, P. Fontaine),	(pred_instance, Profession),	(obj_instance, Musician)}
e_2	`true`	{(source_tableID, tbl #1),	(subj_instance, P. Fontaine),	(pred_instance, DoB),	(obj_instance, c.1380)}
e_3	`true`	{(source_tableID, tbl #1),	(subj_instance, P. Fontaine),	(pred_instance, DoD),	(obj_instance, c.1450)}
e_4	`true`	{(source_tableID, tbl #1),	(subj_instance, J. Vide),	(pred_instance, Profession),	(obj_instance, Musician)}
e_5	`false`	{(source_tableID, tbl #1),	(subj_instance, J. Vide),	(pred_instance, DoB),	(obj_instance, 01/01/1900)}
...	...	...	...	...	...

(a) List of Elements: The triples of Figure 1b, represented in the element format. The truthfulness value (V) of incorrect elements is `false`.

	Feature	Property vector	Structure vector	List of elements
Level 0	f_0	{ALL, ALL, ALL, ALL}	{0, 0, 0, 0}	$\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$
Level 1	f_1	{(source_URL, *wiki*), ALL, ALL, ALL}	{1, 0, 0, 0}	$\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$
	f_2	{ALL, (subj_type, *People*), ALL,ALL}	{0, 1, 0, 0}	$\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$
	f_3	{ALL, ALL, (pred_type, *Bio*), ALL}	{0, 0, 1, 0}	$\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$
	f_4	{ALL, ALL, ALL, (obj_type, *Profession*)}	{0, 0, 0, 1}	$\{e_1, e_4, e_7, e_{10}\}$
	f_5	{ALL, ALL, ALL, (obj_type, *Date*)}	{0, 0, 0, 1}	$\{e_2, e_3, e_5, e_6, e_8, e_9, e_{11}, e_{12}\}$
Level 2	f_6	{ALL, ALL, ALL, (obj_instance, *01/01/1900*)}	{0, 0, 0, 2}	$\{e_5, e_9, e_{12}\}$
	f_7	{ALL, ALL, ALL, (obj_instance, *c.1380*)}	{0, 0, 0, 2}	$\{e_2\}$
...	...	...	...	...
Level 4	f_8	{(source_tableID, *tbl #1*), ALL, (pred_instance, *DoB*), ALL }	{2, 0, 2, 0}	$\{e_2, e_5\}$
	f_9	{(source_tableID, *tbl #2*), ALL, (pred_instance, *DoD*), ALL }	{2, 0, 2, 0}	$\{e_9, e_{12}\}$
...	...	...	...	...

(b) List of Features: Candidate reasons of extraction errors in the feature format. The incorrect elements in each feature are marked as red. *Structure vector* and *feature level* are defined in Section 4.

Figure 2: The Element-Feature model is a more efficient representation of the data instance and the possible error causes that takes advantage of the hierarchical relationship of the extraction properties.

DEFINITION 7 (OPTIMAL DIAGNOSIS). *Given a dataset of elements $\mathcal{E} = \{e_1, ..., e_n\}$ and a cost function c, the optimal diagnosis is the set of features, $\mathcal{F} = \{f_1, ..., f_k\}$, such that*

- *$\forall e_i \in \mathcal{E}$ with $e_i.V =$ `false`, $\exists f_j \in \mathcal{F}$ such that $e_i \in f_j.\mathbf{E}$;*
- *$c(\mathcal{F})$ is minimized.*

The first condition of the definition requires that the diagnosis explains *all* errors, while the second condition requires that the diagnosis is optimal with respect to a cost function c. Note again that the output is a set of features that are associated with the causes, instead of the causes themselves.

In the rest of the paper we tackle two challenges: (1) how to derive an appropriate cost function, and (2) how to solve the optimal diagnosis problem efficiently.

Overview: cost model (Section 3).

We start by using Bayesian analysis to derive the set of features with the highest probability of being associated with the causes for the mistakes in the dataset. We derive our cost function from the Bayesian estimate: the lowest cost corresponds to the highest *a posteriori* probability that the selected features are the real causes for the errors. The resulting cost function contains three types of penalties, which capture the following three intuitions:

Conciseness: Simpler diagnoses with fewer features are preferable.
Specificity: Each feature should have a high error rate.
Consistency: Diagnoses should not include many correct elements.

We propose an additive cost function based on these three penalties that approximates the Bayesian estimate and is efficient to compute.

Overview: diagnostic algorithm (Section 4).

We can prove that finding the set of features with the minimal cost is NP-complete. We design a top-down, iterative algorithm with linear-time complexity. Our algorithm traverses the feature hierarchy from coarser to finer granularity features. It uses local stopping conditions to decide whether to accept the current feature or explore deeper. We extend our algorithm to a parallel, MapReduce-based version that is effective at large-scale diagnosis tasks.

3. DIAGNOSTIC FRAMEWORK

The core of our diagnostic framework is the cost model used to determine the optimal diagnosis (Definition 7). In this section, we focus on deriving a cost function that is effective in identifying good diagnoses and that can be computed efficiently. We start by using Bayesian analysis to compute the probability that a set of features is the cause of the mistakes in the dataset (Section 3.1). Then, we propose a cost function that approximates the Bayesian analysis efficiently, through simple, additive penalty functions (Section 3.2). Finally, we show that deriving the optimal diagnosis is NP-complete, which further motivates the need for efficient algorithms with parallelization potential (Section 3.3).

3.1 Bayesian estimate of causal likelihood

Given a set of elements $\mathcal{E} = \{e_1, \ldots, e_n\}$ and their correctness, we wish to estimate the probability $\Pr(\mathcal{F}|\mathcal{E})$ that a set of features $\mathcal{F} = \{f_1, \ldots, f_k\}$ is the cause for the incorrect data instances in $\mathcal{E}$. From Bayesian inference, the *a posteriori* probability $\Pr(\mathcal{F}|\mathcal{E})$ is proportional to the likelihood $\Pr(\mathcal{E}|\mathcal{F})$ times the prior $\Pr(\mathcal{F})$:

$$\Pr(\mathcal{F}|\mathcal{E}) \propto \Pr(\mathcal{E}|\mathcal{F}) \Pr(\mathcal{F}) \quad (1)$$

We assume that mistakes represented by features are independent. This assumption is reasonable because even for related features, the associated causes can still be independent. For example, feature $f_6 = \{\text{ALL}, \text{ALL}, \text{ALL}, (obj\_instance, 01/01/1900)\}$ is subsumed by feature $f_5 = \{\text{ALL}, \text{ALL}, \text{ALL}, (obj\_type, Date)\}$; however, f_6 can be associated with the cause of incorrectly assigning the default value *01/01/1900*, while f_5 can be associated with

the cause of inability of an extractor to parse dates, and the two causes are independent. Using this independence assumption, we can express the prior $\Pr(\mathcal{F})$ as follows.

$$\Pr(\mathcal{F}) = \prod_{f_i \in \mathcal{F}} \Pr(f_i) \tag{2}$$

We further use α to denote the *a priori* probability that a feature is a cause ($\Pr(f_i) = \alpha$).[1] Then, $\Pr(\mathcal{F}) = \alpha^k$.

Now consider $\Pr(\mathcal{E}|\mathcal{F})$: We assume the elements in $\mathcal{E}$ are independent conditioned on $\mathcal{F}$. For an element $e_j \in \mathcal{E}$, we denote by $F(e_j) \subseteq \mathcal{F}$ the features that contain e_j; only errors associated with features in $F(e_j)$ can affect the correctness of e_j. Thus, we have the following.

$$\Pr(\mathcal{E}|\mathcal{F}) = \prod_{e_j \in \mathcal{E}} \Pr(e_j|\mathcal{F}) = \prod_{e_j \in \mathcal{E}} \Pr(e_j|F(e_j)) \tag{3}$$

We assume that for each cause of errors, there is an error rate between 0 and 1. For example, assigning a default date *01/01/1900* often has an error rate close to 1 (if not 1), date format parsing error from a particular webpage can also have a high error rate, whereas a webtable providing erroneous data often has a lower error rate. We denote by ϵ_i the error rate of the cause associated with feature f_i. The error rate ϵ_i can be derived directly from $f_i.\mathbf{E}$ and denotes the probability that an element represented by feature f_i is incorrect when f_i is associated with a cause of error.

Then, the probability of an element e_j being correct is the probability that none of the causes associated with the features it belongs to affects its correctness. Similarly, we can compute the probability of an element being incorrect.

$$\Pr(e_j.V = \text{true}|F(e_j)) = \Pi_{f_i \in F(e_j)}(1 - \epsilon_i) \tag{4}$$

$$\Pr(e_j.V = \text{false}|F(e_j)) = 1 - \Pi_{f_i \in F(e_j)}(1 - \epsilon_i) \tag{5}$$

As special cases, we define $Pr(e_j.V = \text{true}|\emptyset) = 1$, rewarding not including correct elements in the returned features, and define $Pr(e_j.V = \text{false}|\emptyset) = 0$, penalizing not including incorrect elements in the returned features. Since Definition 7 requires covering all incorrect elements, we assume in the rest of the paper that $F(e_i) \neq \emptyset$ for every $e_i \in \mathcal{E}, e_i.V = \text{false}$.

Equations (1–5) together compute the probability that the features in set $\mathcal{F}$ are the causes of the errors in data instances of $\mathcal{E}$. Our goal is to find the set $\mathcal{F}$ with the highest probability $\Pr(\mathcal{F}|\mathcal{E})$.

EXAMPLE 8. *We consider two sets of features, $\mathcal{F}_1 = \{f_6\}$ and $\mathcal{F}_2 = \{f_8, f_9\}$, as possible causes for the errors in the set of elements $\mathcal{E} = \{e_1, \ldots, e_{12}\}$ in Figure 2a. Semantically, the former means that the errors are caused by the wrong default value "01/01/1900"; the latter means that the errors are caused by two mistakes: wrongly parsing the dates of birth in Table 1 and wrongly parsing the dates of death in Table 2.*

Feature f_6 has 3 incorrect elements and no correct elements; its error rate is $\epsilon_6 = 1$. Using $\alpha = 0.5$, we get that $Pr(\mathcal{F}_1) = 0.5$, $Pr(\mathcal{E}|\mathcal{F}_1) = 1 - (1 - 1) = 1$, so $Pr(\mathcal{F}_1|\mathcal{E}) \propto 0.5$.

On the other hand, f_8 has one incorrect element, and one correct element ($\epsilon_8 = 0.5$), and f_9 has two incorrect elements and no correct elements ($\epsilon_9 = 1$). Thus, $Pr(\mathcal{F}_2) = 0.5^2$, $Pr(e_2|\mathcal{F}_2) = Pr(e_2|f_8) = (1 - 0.5) = 0.5$, $Pr(e_5|\mathcal{F}_2) = Pr(e_5|f_8) = 1 - (1 - 0.5) = 0.5$, $Pr(e_9|\mathcal{F}_2) = Pr(e_{12}|\mathcal{F}_2) = 1 - (1 - 1) = 1$, so $Pr(\mathcal{F}_2|\mathcal{E}) \propto 0.5^2 \cdot 0.5 \cdot 0.5 \cdot 1 \cdot 1 = 0.0625$. This result indicates that $\mathcal{F}_1$ is more likely to be the cause of the errors than $\mathcal{F}_2$.

[1]For simplicity, we assume that all features have the same *a priori* probability of failure. However, this is not imperative in our model, and different probabilities can be used.

3.2 The diagnostic cost model

The Bayesian analysis described previously represents the probability that a set of features $\mathcal{F}$ are the causes of the errors in $\mathcal{E}$. It requires probability computation for each element. Since our goal is to find the set of features that best diagnose the errors, it would be much more intuitive to transform the *a posteriori* probability to a cost function that computes a cost for each feature, and sums up the cost for all selected features.

Note that both $\Pr(\mathcal{F})$ and $\Pr(e_j.V = \text{true}|F(e_j))$ can be written as the product of a set of terms, each associated with a feature. If we can transform $\Pr(e_j.V = \text{false}|F(e_j))$ to such a form, we can then define a cost function for each feature and take an aggregation. For this purpose, we estimate $\Pr(e_j.V = \text{false}|F(e_j))$ as follows.

$$\Pr(e_j.V = \text{false}|F(e_j)) = \Pi_{f_i \in F(e_j)} \epsilon_i \tag{6}$$

In general, this estimate can be arbitrarily bad (consider the extreme case $|F(e_j)| \to \infty$). However, in practice, an erroneous element is usually due to one or two reasons instead of many. Therefore, ideally, $|F(e_j)|$ should be small in the diagnosis (i.e., little overlap between returned features). Our estimate is precise when $|F(e_j)| = 1$. Furthermore, when $|F(e_j)| > 1$, Equation (6) computes a lower probability than (5), penalizing overlapping features even hasher, which is consistent with our intuition. Our experimental results verify that this estimate leads to good diagnosis results.

We combine Equations (1–4) and (6) to get the following expression for the probability $\Pr(\mathcal{F}|\mathcal{E})$.

$$\begin{aligned}\Pr(\mathcal{F}|\mathcal{E}) &\propto \prod_{f_i \in \mathcal{F}} \Pr(f_i) \prod_{e_j \in \mathcal{E}} \Pr(e_j|F(e_j)) \\ &= \prod_{f_i \in \mathcal{F}} \alpha \epsilon_i^{|f_i.\mathbf{E}_i^-|} (1 - \epsilon_i)^{|f_i.\mathbf{E}_i^+|}\end{aligned} \tag{7}$$

where $f_i.\mathbf{E}^-$ and $f_i.\mathbf{E}^+$ are the sets of false and true elements of f_i, respectively. Equation (7) contains three distinct components that comprise the probability $\Pr(\mathcal{F}|\mathcal{E})$: a fixed factor ($\alpha$), a factor that corresponds to false elements of the feature, and a factor that corresponds to true elements. Accordingly, we define a cost function for each feature.

DEFINITION 9 (FEATURE COST). *The cost $c(f_i)$ of a feature f_i is the sum of the* fixed cost*,* false cost*, and* true cost*, defined as:*

$$c_{fixed}(f_i) = \log \frac{1}{\alpha}$$

$$c_{false}(f_i) = |f_i.\mathbf{E}^-| \log \frac{1}{\epsilon_i}$$

$$c_{true}(f_i) = |f_i.\mathbf{E}^+| \log \frac{1}{1 - \epsilon_i}$$

The use of logarithms[2] allows our cost function to be additive. Then, the diagnosis cost, which we define below, is logarithmically proportional to the *a posteriori* probability $\Pr(\mathcal{F}|\mathcal{E})$ in Equation (7).

DEFINITION 10 (DIAGNOSIS COST). *The cost $c(\mathcal{F})$ of a diagnosis $\mathcal{F} = \{F_1, ..., F_k\}$ is the sum of the costs of all its features:*

$$c(\mathcal{F}) = \sum_{f_i \in \mathcal{F}} c(f_i)$$

EXAMPLE 11. *We revisit the candidate diagnoses of Example 8: $\mathcal{F}_1 = \{f_6\}$ and $\mathcal{F}_2 = \{f_8, f_9\}$. The costs of the relevant features*

[2]Without loss of generality, we assume logarithms of base 2.

are: $c(f_6) = c_{fixed}(f_6) + c_{false}(f_6) + c_{true}(f_6) = 1 + 1 \cdot 0 + 0 = 1$, $c(f_8) = 1 + 1 \cdot 1 + 1 \cdot 1 = 3$, *and* $c(f_9) = 1 + 2 \cdot 0 + 0 = 1$. *Therefore,* $c(\mathcal{F}_1) = 1$ *and* $c(\mathcal{F}_2) = 3 + 1 = 4$. *Since* $\mathcal{F}_1$ *has a lower cost, it is a better diagnosis than* $\mathcal{F}_2$.

Note also that both $\mathcal{F}_1$ *and* $\mathcal{F}_2$ *contain only disjoint features, so their costs estimate the corresponding probabilities precisely:* $\Pr(\mathcal{F}_1|\mathcal{E}) = 2^{-c(\mathcal{F}_1)} = 0.5$ *and* $\Pr(\mathcal{F}_2|\mathcal{E}) = 2^{-c(\mathcal{F}_2)} = 0.0625$.

Interestingly, the three penalties considered for the feature cost capture the three important properties for the returned diagnosis.

Conciseness: Penalty $c_{fixed}(f_i) > 0$ represents the *a priori* probability. This means that the diagnosis cost increases as the size of $\mathcal{F}$ increases, so this factor prioritizes smaller feature sets (i.e., concise explanations).

Specificity: Penalty $c_{false}(f_i)$ prioritizes the choice of features with higher error rate to cover the same wrong element. If two features cover the same wrong elements, the one with higher error rate will result in a lower cost.

Consistency: Penalty $c_{true}(f_i)$ prioritizes the choice of features that contain fewer correct elements. Feature sets that cover a lot of correct elements will result in a high cost.

Adding these cost penalties balances conciseness, specificity, and consistency. For example, the diagnosis with a single feature that contains all elements is obviously the most concise diagnosis, but its error rate is presumably low and it involves a lot of correct elements, so there is a high true cost and false cost. On the other hand, returning each element as a single-element feature in the diagnosis is obviously the most specific and consistent diagnosis, but the number of features is high, resulting in a high fixed cost.

3.3 Complexity

For a given dataset of elements $\mathcal{E}$, our cost model assigns a constant cost to each feature. This transforms the problem of deriving optimal diagnoses (Definition 7) to a weighted set cover problem [29]. There is a straightforward reduction from weighted set cover to the problem of optimal diagnosis, which means that our problem is NP-complete.

THEOREM 12 (NP-COMPLETENESS). *Given a dataset of elements* $\mathcal{E} = \{e_1, ..., e_n\}$*, the cost function* c *of Definition 10, and a maximum cost* K*, determining whether there exists a diagnosis* $\mathcal{F}$*, such that* $c(\mathcal{F}) \leq K$*, is NP-complete.*

Weighted set cover is a well established problem, with extensive related work. Specifically, there are several approximation algorithms for this problem [13, 16, 27], but typically, they don't come near to addressing the scale of problems that are relevant to our motivating application domain. These algorithms typically have high-degree polynomial complexity (e.g., quadratic in the number of features [27]), and they are not amenable to parallelism.

In the next section, we introduce a powerful, sort-free, top-down iterative algorithm for the optimal diagnosis problem, with *linear time complexity* and great parallelization potential. We extend our algorithm to a MapReduce-based implementation, and show that it is both effective and efficient.

4. DERIVING A DIAGNOSIS

In this section, we propose an algorithm that can derive diagnoses efficiently, by exploiting the hierarchical structure of features. We start with a description of the feature hierarchy (Section 4.1). We then propose an algorithm that constructs a diagnosis by traversing the hierarchy in a top-down fashion (Section 4.2). Finally, we present a MapReduce version of our algorithm that makes causal diagnosis practical for large-scale datasets (Section 4.3).

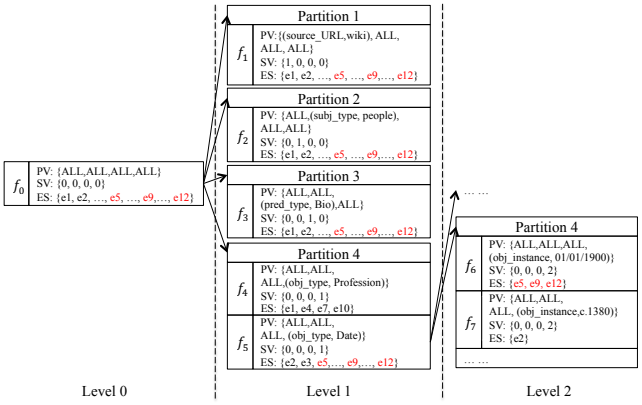


Figure 3: The hierarchy structure of the features of Figure 2b.

4.1 The feature hierarchy

Features form a natural hierarchy due to the property hierarchy (Section 2.1). For example, the feature {(source_URL, wiki), ALL, ALL, ALL} contains the feature {(source_tableID, tbl #1), ALL, ALL, ALL}; semantically, the table is a subset of the wiki page. This hierarchy is embedded in the features' property vectors, and we model it explicitly by deriving a feature's *structure vector* and its hierarchy *level*.

Structure vector (SV): A SV is an integer vector $\{s_1, \ldots, s_m\}$, where the i-th element represents the granularity of the feature in the i-th property dimension. Lower numbers represent coarser granularity, with 0 mapping to Root. For example, in the source dimension, we have the granularity levels Root, source_URL, and source_tableID, which are represented in the structure vector with 0, 1, and 2, respectively. Therefore, the structure vector of feature f_1 in Figure 2b is $\{1, 0, 0, 0\}$, because *wiki* is a value at the granularity level of source_URL. The structure vector is derived from a feature's property vector, and provides an intuitive representation of the feature's relationship with other features.

Feature level: The feature level is an integer that denotes the distance of a feature from the root of the feature hierarchy. It can be computed directly from the structure vector, as the sum of all its dimensions ($\sum_i s_i$). For example, feature f_1 has level 1.

Feature hierarchy: We define the parent-child relationships in the feature hierarchy using the list of feature elements ($f.\mathbf{E}$), the feature structure vector (SV_f) and the feature level (L_f).

DEFINITION 13 (PARENT-CHILD FEATURES). *A feature* f_p *is the parent of feature* f_c *(equivalently,* f_c *is a child of* f_p*) when the following conditions hold:*

(a) $e \in f_c.\mathbf{E} \Rightarrow e \in f_p.\mathbf{E}$

(b) $L_{f_p} = L_{f_c} - 1$

(c) $\forall i \in [1, m]$, $SV_{f_p}(i) \geq SV_{f_c}(i)$

The conditions of the definition ensure that (a) the elements of a parent feature are a superset of the elements of the child feature, (b) the parent feature is one level closer to the root of the hierarchy, and (c) each dimension of the child feature has equal or finer granularity than the parent.

A feature can have multiple parents. For example, the features {(source_URL, wiki), ALL, (pred_instance, DoB), ALL} and {(source_tableID, tbl #1), ALL, (pred_type, Bio), ALL} are both parents of feature {(source_tableID, tbl #1), ALL, (pred_instance, DoB),

Algorithm 1 DATAXRAY

Require: A set of elements $\mathcal{E}$;
Ensure: A set of problematic features $\mathbf{R}$;
1: $parentList \leftarrow$ InitialFeature($elementList$);
2: $R \leftarrow \emptyset$;
3: **while** $parentList! = \emptyset$ **do**
4: $S, U, childList, nextLevel \leftarrow \emptyset$;
5: **for** each $parentF \in parentList$ **do**
6: SPLITFEATURE($parentF, childList$);
7: **end for**
8: $partitionList \leftarrow$ getPartition($parentList, childList$);
9: **for** each $partition \in partitionList$ **do**
10: COMPAREFEATURE($partition, S, U$);
11: **end for**
12: MERGEFEATURE($parentList, nextLevel, S, U, R$);
13: $parentList \leftarrow nextLevel$;
14: **end while**
15: **return** R

ALL}. The hierarchy defined by the parent-child feature relationships is a directed acyclic graph (DAG). The root of the hierarchy is feature {ALL, ALL, ALL, ALL}, and each leaf is a feature that maps to a unique element. For example, the feature {(source_tableID, tbl #1), (subj_instance, J. Vide), (pred_instance, DoB), (obj_instance, 01/01/1900)}, at level 8, represents element e_5 in Figure 2a, and equivalently, triple t_5 in Figure 1b.

Feature partitions: Features at the same hierarchy level generally have overlapping sets of elements. For example, f_1 and f_4 have four elements in common. This can be a problem for an algorithm that explores the hierarchy, because it is harder to compare features that overlap in an arbitrary way. To address this problem, we organize the child features of a parent feature f_p into m *partitions*, where m is the number of property dimensions.

DEFINITION 14 (PARTITION). *A partition $\mathcal{P}_i^{f_p}$ contains every feature f that is a child feature of f_p and $SV_{f_p}(i) = SV_f(i) - 1$.*

For example, f_4 and f_5 form partition $\mathcal{P}_4^{f_0}$ in level 1: they share the same parent, f_0, and $SV_{f_4}(4) = SV_{f_5}(4) = SV_{f_0}(4) + 1$. Overall, level 1 has four partitions: $\mathcal{P}_1^{f_0} = \{f_1\}$, $\mathcal{P}_2^{f_0} = \{f_2\}$, $\mathcal{P}_3^{f_0} = \{f_3\}$, and $\mathcal{P}_4^{f_0} = \{f_4, f_5\}$. By construction, partitions ensure that their features do not overlap (e.g., $f_4.\mathbf{E} \cap f_5.\mathbf{E} = \emptyset$), and the union of all their features cover all the parent elements (e.g., $f_4.\mathbf{E} \cup f_5.\mathbf{E} = f_0.\mathbf{E}$).

4.2 Top-down iterative diagnosis

Our diagnostic algorithm traverses the hierarchy in a top-down fashion (breadth-first), exploring coarse-granularity features first, and drilling down to finer-granularity features while improving the cost estimate at every step. The algorithm maintains three sets of features in the traversal.

- *Unlikely causes* $\mathbf{U}$*:* features that are not likely to be causes.
- *Suspect causes* $\mathbf{S}$*:* features that are possibly the causes.
- *Result diagnosis* $\mathbf{R}$*:* features that are decided to be associated with the causes.

Our algorithm, DATAXRAY, is described in Algorithm 1. At a high level, every iteration of the algorithm considers features at a particular level (Line 5–Line 7), compares each parent feature with its child features, and populates the list of suspect causes $\mathbf{S}$ and the list of unlikely causes $\mathbf{U}$ (Line 8–Line 11). At the end of the iteration, the sets $\mathbf{S}$ and $\mathbf{U}$ are consolidated (Line 12): parent features that occur only in $\mathbf{S}$ are added to the result diagnosis $\mathbf{R}$, and all elements that $\mathbf{R}$ contains are marked as "covered"; child features that occur only in $\mathbf{S}$ are kept for traversal in the next iteration. The traversal completes once all incorrect elements are marked as being covered by some feature in $\mathbf{R}$.

We next describe the major components of the algorithm.

SPLITFEATURE: Given a feature and its elements, this component derives the corresponding child features and partitions. It uses the structure vector of the parent feature to derive the structure vectors of the children and the partition of the parent. It then generates the property vectors of each child feature by examining the elements in the parent. Finally, if the parent feature is marked as "covered", all the elements of the feature are already covered by ancestor features selected to $\mathbf{R}$. To avoid producing redundant diagnoses, the child features of the current feature are also marked as "covered". Features marked as "covered" will not be added to $\mathbf{R}$.

COMPAREFEATURE: Given a parent node and a partition, this component compares the feature set containing only the parent, and the feature set containing all child features in the partition, to determine which is a better solution. The winner features are added to $\mathbf{S}$ and the loser features are added to $\mathbf{U}$. The comparison is based on the cost model of Definition 10. In Section 4.2.1 we describe two additional criteria that simplify this computation.

MERGEFEATURE: In the previous step, each partition populates the sets $\mathbf{S}$ and $\mathbf{U}$ independently. This component consolidates the results. Parent features only in $\mathbf{S}$ and not in $\mathbf{U}$ transfer to the result diagnosis $\mathbf{R}$, and their child features are marked as "covered". Parent features in $\mathbf{U}$ are discarded, since it means there exists some partition where the child features form a better solution. Child features in $\mathbf{S}$ are sent to the next iteration for further traversal.

THEOREM 15 (COMPLEXITY AND BOUNDS). *Algorithm 1 has complexity $O(|\mathcal{F}|)$, and provides an $O(n)$-approximation of the minimum cost diagnosis, where n is the number of elements in $\mathcal{E}$ and $\mathcal{F}$ is the set of features that can be derived from $\mathcal{E}$.*

DATAXRAY provides an $O(n)$-approximation of the optimal diagnosis, which is worse than the greedy approximation bound for set cover ($O(\log n)$). However, this worst case setting requires that errors are uniformly distributed among the child nodes of a feature, *across all dimensions*. This is extremely unusual in practice, and in most cases DATAXRAY significantly outperforms approximations for set cover.

In Section 5, we show that DATAXRAY outperforms greedy set cover by an order of magnitude. Our algorithm exploits the hierarchy of the features, leading to better worst-case complexity (linear in the number of features) than other approximations for set cover. We note that the number of features can be huge: $O(l^m|\mathcal{E}|)$, where m is the number of dimensions and l the maximum number of levels in the property hierarchy for each dimension. However, in practice, m is usually small. Moreover, DATAXRAY is by design highly-parallelizable and we show how to implement it in the MapReduce framework (Section 4.3).

4.2.1 Optimizations

Line 10 of Algorithm 1 compares two sets of features using the cost model of Definition 10. This computation requires enumerating each element in the feature sets. We use two heuristic criteria that simplify computation and prune features faster.

Variance pruning: The *variance* of a feature describes how the errors are distributed among the child features. We compute the

variance in each partition $\mathcal{P}_i^f$ of a feature f as:

$$Var_i^f = \sum_{f_c \in \mathcal{P}_i^f} \frac{(\epsilon_{f_c} - \epsilon_f)^2}{|\mathcal{P}_i^f|}$$

Intuitively, if a feature is associated with a cause of errors, it is likely to result in uniform mistakes across its child features, resulting in low variance of error rates among its children. A feature with high variance indicates that the actual culprit is deeper in the feature hierarchy; we thus add a parent feature f to $\mathbf{U}$ if $Var_i^f \geq \theta_{max}$. Based on empirical testing, we chose $\theta_{max} = 0.1$ for our experiments. For example, feature f_5 in Figure 2b has 6 child features in partition $\mathcal{P}_4^{f_5}$; five with zero error rate, and one with $\epsilon = 1$. Then, the variance in that partition is $Var_4^{f_6} = 0.14 > \theta_{max}$, so f_6 is added to $\mathbf{U}$.

Error rate pruning. When a feature is associated with a cause of errors, typically its error rate would not be too low. Accordingly, we add a parent feature f to $\mathbf{U}$ if $\epsilon_f \leq \delta_{min}$. Again, empirically, we chose $\delta_{min} = 0.6$.

4.2.2 Greedy refinement

Our diagnostic framework does not consider correlations among features. If correlations exist, they can result in diagnoses that contain redundant features (i.e., features with a lot of overlap). DATAXRAY detects redundancies across features of different levels, but is unaware of overlap in features selected from the same hierarchy level. To eliminate such redundancies in the resulting diagnosis, we post-process it with a greedy set-cover step. This greedy step looks for a minimal set of features among those chosen by DATAXRAY. Since the number of features in the DATAXRAY result is typically small, this step is very efficient. In Section 5, we show that with negligible overhead, DATAXRAY with greedy refinement results in significant improvements in accuracy.

4.3 Parallel diagnosis in MapReduce

We design our algorithm with parallelization in mind: the split, compare, and merge steps can each execute in parallel, as the computation always focuses on a specific partition or feature. In this section, we describe how our algorithm works in a MapReduce framework, creating a separate map-reduce stage for each of the split, compare, and merge functions.

Stage I parallelizes the generation of child features. The *Map* phase maps each element in the parent feature to relevant child features; in other words, for each element in a parent feature, it generates pairs where the element is the value and a child property vector is the key. The *Reduce* phase generates each child feature according to the set of elements, computes its error rate and cost.

Stage II parallelizes the comparison of each parent feature and a partition of child features. The *Map* phase generates, for each child feature, the partitions it belongs to; in other words, for each partition that contains the child feature, it generates a pair where the child is the value and the parent-partition pair is the key. The *Reduce* phase compares the parent feature with each partition of its child features.

Stage III parallelizes the decision of whether to discard a feature, or to return a feature in the diagnosis, or to keep it for further traversal. The *Map* phase populates $\mathbf{S}$ and $\mathbf{U}$; in other words, for each feature in the comparison, it generates a pair where the feature is the key and the decision for adding it to $\mathbf{S}$ or $\mathbf{U}$ is the value. The *Reduce* phase makes a final decision for each feature.

5. EXPERIMENTAL EVALUATION

This section describes a thorough evaluation of our diagnostic framework on real-world knowledge extraction data, as well as large-scale synthetic data. Our results show that (1) our cost function models the quality of diagnoses effectively; (2) our algorithm is both more effective and more efficient than other existing techniques; and (3) our MapReduce implementation of the algorithm is effective at handling datasets of large scale.

Extraction	false triples	true triples	error rate
$reverb$	304	315	0.49
$reverbnolex$	338	290	0.54
$textrunner$	478	203	0.70
woe^{pos}	535	218	0.71
woe^{parse}	557	324	0.63

Figure 4: Real-world datasets from 5 different knowledge extraction systems of the ReVerb ClueWeb Extraction dataset [24].

Datasets

We first describe the real-world data used in our evaluation; we describe our synthetic data experiments in Section 5.2.

Knowledge triple extraction systems. We demonstrate the effectiveness of our diagnosis framework in practice, using five real-world knowledge extraction systems of the ReVerb ClueWeb Extraction dataset [24]. Figure 4 provides high-level characteristics about each of these 5 extractors. The dataset samples 500 sentences from the web, using Yahoo!'s random link service. The dataset contains labeled knowledge triples: each triple has a `true` or `false` label indicating whether it is correct or incorrect, respectively.

We proceed to describe how we model the knowledge extraction datasets in our feature-based framework. In our model, each knowledge triple is an element with a 5-dimensional property vector, with the following property hierarchies:

1. Source (Root, sentenceID) describes which sentence the triple is extracted from.

2–4. Subject, Predicate, Object (Root, structure, content). Each of these dimensions describes the structure of the sentence, and the content value. The structure is composed by the *Pos Tags* [55] (e.g., *none, verb*). The content is the actual content value of the triple.

5. Confidence (Root, confidence bucket). Extraction systems annotate the extracted knowledge with confidence values as an assessment of their quality. We capture the confidence bucket as part of the property dimensions.

In our experiments, we focused on these 5 dimensions, because of limited knowledge of each systems' inner workings. In practice, domain experts are likely to include more dimensions (e.g., specific extraction patterns) to derive more accurate diagnoses.

Silver standard. The dataset does not provide a "gold standard" of diagnoses for the erroneous triples. We manually derived a "silver standard" against which we evaluate our methods. In particular, we considered every feature returned by each alternative technique we implemented, and manually investigated if it is very likely to be associated with a particular error. To the best of our knowledge, there is no freely available dataset with labeled diagnoses; our manually derived silver standard is the best-effort approach to evaluating our techniques while grounded to real-world data.

Comparisons

We compare two versions of our algorithms, DATAXRAY and DATAXRAY+GREEDY with several alternative algorithms and state-of-the-art methods designed for similar problem settings: a greedy algorithm for set cover, GREEDY, and a variant with different op-

timization objective, REDBLUE; a data quality exploration tool, DATAAUDITOR; and two classification algorithms, FEATURESELECTION and DECISIONTREE.

DATAXRAY (Section 4): Derives diagnoses by identifying "bad" features using the DATAXRAY algorithm proposed in this paper. We set $\alpha = 0.1$, used in the fixed cost, and $\theta_{max} = 0.1$ and $\delta_{min} = 0.6$, used in the pruning and filtering heuristics.

DATAXRAY+GREEDY (Section 4.2.2): This algorithm applies a greedy set-cover refinement step on the result of DATAXRAY to eliminate redundancies.

GREEDY [13]: We apply the greedy approximation for weighted set cover to select the set of features of minimum cost that cover all of the `false` elements, according to our cost model (Section 3.2). Our cost model allows set cover to penalize features that cover `true` elements, which it does not do in its default objective.

REDBLUE [10,48]: Given a collection of sets with "blue" and "red" elements, the red-blue set cover problem looks for a sub-collection of sets that covers all "blue" elements and minimum number of "red" elements. In contrast to regular set-cover, red-blue set cover can model both correct and incorrect element coverage. We map `false` elements to "blue" elements, and `true` elements to "red" elements, while features are sets. We use a greedy approximation algorithm [48] to find the cover.

DATAAUDITOR [30, 31]: We use Data Auditor, a data quality exploration tool that uses rules and integrity constraints to construct *pattern tableaux*. We annotate false elements as a consequent (dependent) value in a FD, and use Data Auditor to learn a pattern for this rule, which we treat as a diagnosis. We set support $\hat{s} = 1$ to diagnose all false elements; the confidence $\hat{c}$ corresponds to the error rate pruning in DATAXRAY, so we set $\hat{c} = \delta_{min} = 0.6$.

FEATURESELECTION [46, 54]: We use logistic regression to derive a set of features that is a good classifier between `true` and `false` elements. For each feature the algorithm learns a weight between -1 and 1: a positive weight indicates that the feature is positive proportional to the class (in our context the feature is a cause), and a negative weight indicates the opposite. We use our labeled data as the training dataset, excluding features with only `true` elements to speed up learning, and return features with positive weights. We apply L_1-regularization, which favors fewer features for the purpose of avoiding over-fitting. We use 0.01 as the regularization parameter (a higher parameter applies a higher penalty for including more features), as we empirically found that it gives the best results.

DECISIONTREE [**?**]: We use decision trees with pruning as an alternative classification method. Pruning avoids overfitting, which would lead decision trees to always select features at the lowest hierarchy levels. We set low confidence (0.25) to promote pruning, and restrict the minimum number of instances to two (each selected feature should have at least two elements). We found empirically that these parameters provide the best results.

We use the SLEP package implementation for logistic regression [38] and WEKA [**?**] for decision trees, and we implemented the rest of the algorithms in Java. In addition, the MapReduce version of DATAXRAY uses Hadoop APIs.

Metrics

We evaluate the effectiveness and efficiency of the methods.

Precision/Recall/F-measure: We measure the correctness of the derived diagnoses using three measures: *Precision* measures the portion of features that are correctly identified as part of the optimal diagnosis; *Recall* measures the portion of features associated with causes of errors that appear in the derived diagnosis; *F-measure* computes their harmonic mean ($\frac{2 \cdot precision \cdot recall}{precision + recall}$). Note that we do not know all causes for the errors, so recall is evaluated against the union of all features marked as correct diagnoses in our silver standard.

Execution time: We report the execution time for each method, broken down into preprocessing time (prep.), computation time (comp.), and total execution time (total time). The preprocessing time (Prep.) for DATAXRAY and DATAXRAY+GREEDY is the time to compose the initial root feature at level 0. For the other methods (GREEDY, REDBLUE, and FEATURESELECTION), the preprocessing time accounts for the time to find all eligible features and compute their costs if necessary. Computation time (Comp.) is the time that each method takes on average, after the preprocessing step. The total time is the sum of preprocessing and computation time.

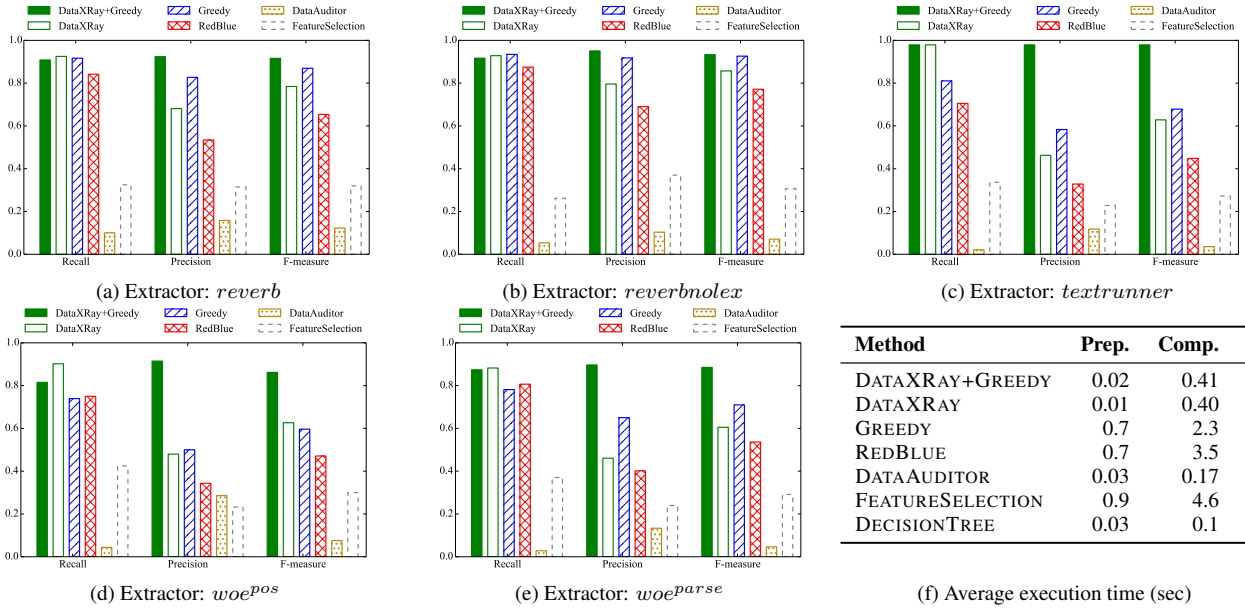
We ran experiments comparing all of the methods on an iMac with 3.2 GHz Intel Core i5 processor, 8GB RAM. We also conducted experiments on the scalability of our MapReduce implementation. The experiments were conducted on a Hadoop 2.2.0 cluster with 15 slave nodes. The head node has 2.4 GHz processor and 24GB RAM. The slave nodes have 2.66 GHz processor and 16GB RAM.

5.1 Real-world data

In our first round of experiments, we test our diagnostic framework using real-world knowledge extraction data. Figures 5a–5e report the quality of the diagnoses produced by each method on the data extracted by five real-world extraction systems. Figure 5f reports the average execution time for each method. Our results in Figure 5 demonstrate that our framework derives better diagnoses than the other approaches, and does so more efficiently.

Our first goal is to evaluate the effectiveness of our cost function. The results in Figures 5a–5e demonstrate that the methods that apply our cost function (DATAXRAY, DATAXRAY+GREEDY, and GREEDY) result in diagnoses of significantly better quality. REDBLUE generally has high recall, but lower precision. This is because REDBLUE favors finer-granularity features: its objective function depends on the number of red elements (i.e., `true` elements) that are included in the diagnosis, but does not consider the number of returned features (size of the diagnosis). DATAAUDITOR also uses a different objective and prioritizes coarse features, leading to bad performance across all extractors. The logistic regression method (FEATURESELECTION) shows low quality in all datasets. The goal of FEATURESELECTION is to build a good prediction model, which is different from our diagnosis goal. Even with L_1-regularization, it may still select small features for the purpose of optimizing classification. We found that the FEATURESELECTION results often contained redundancy and features with low error rates, resulting in both low precision and low recall. DECISIONTREE performs the worst, with F-measure values below 0.015. Compared to FEATURESELECTION, DECISIONTREE is restricted to non-overlapping rules; this reduces the search space, but ignores many of the features, leading to bad performance. These results show that our cost model is successful at producing good diagnoses, and the quality of the diagnoses is significantly better than those produced by methods with alternative objectives.

Our second goal is to evaluate the effectiveness of our approximation algorithms in solving the optimization problem. All of the methods that use our cost model (DATAXRAY, DATAXRAY+GREEDY, and GREEDY) achieve high recall scores in all five datasets. We observe that typically DATAXRAY has a higher recall, whereas GREEDY has a higher precision, especially for the $textrunner$, woe^{pos}, and woe^{parse} datasets. One weakness of DATAXRAY is that it does not detect overlap across features that are selected at the same level of the hierarchy. When that occurs, the resulting diagnoses contain redundancies (multiple features that explain the



(a) Extractor: $reverb$ (b) Extractor: $reverbnolex$ (c) Extractor: $textrunner$

(d) Extractor: woe^{pos} (e) Extractor: woe^{parse}

Method	Prep.	Comp.	Total
DATAXRAY+GREEDY	0.02	0.41	0.43
DATAXRAY	0.01	0.40	0.41
GREEDY	0.7	2.3	3.0
REDBLUE	0.7	3.5	4.2
DATAAUDITOR	0.03	0.17	0.2
FEATURESELECTION	0.9	4.6	5.5
DECISIONTREE	0.03	0.1	0.13

(f) Average execution time (sec)

Figure 5: The quality of the derived diagnoses for all of the methods, across five knowledge extraction systems; the maximum F-measure value achieved by DECISIONTREE was 0.0148, which was too low to display in the graphs. Our approach that combines DATAXRAY with a greedy set-cover step outperforms all other approaches, in some cases significantly. It is also faster than other methods except DATAAUDITOR and DECISIONTREE, but the latter two produce results of very low quality for our problem.

same errors), leading to low precision. DATAXRAY+GREEDY overcomes this weakness by applying the greedy set-cover method over the result of DATAXRAY. This eliminates the redundancy from the DATAXRAY diagnoses, leading to big improvements in precision, and usually with a very small drop on recall. Overall, our DATAXRAY+GREEDY method maintains excellent performance, with F-measure above 0.85 across all five of our datasets.

Our final goal is to evaluate the efficiency of the different algorithms. Figure 5f reports the average execution time for each method. DATAAUDITOR and DECISIONTREE are a bit faster that DATAXRAY on this dataset, but they are not viable for this problem given their terrible F-measure performance. DATAXRAY is an order of magnitude faster than the remaining methods. The greedy refinement is only slightly slower than DATAXRAY: since it executes greedy set cover on the solution of DATAXRAY, the problem space is significantly reduced and thus the greedy step is very efficient.

Interesting diagnoses: Our diagnostic framework identified several interesting problems in these real-world extraction systems. In one case, we found that on the $reverb$ dataset our system produced the feature {ALL, ALL, ALL, (obj_structure, ECC), ALL} as part of a diagnosis, where *ECC* stands for objects ending with coordinating conjunction such as *and, but, for,* and *nor* (e.g., "newspapers and"). This feature corresponds to elements whose objects are extracted from coordinating conjunctions. This indicates a clear problem with the extraction process, as it does not make sense to have a coordinating conjunction as an object in a knowledge triple.

As another example, our method identified {ALL, (subj_structure, CD), ALL, ALL, ALL} as a problem feature in the $textrunner$ dataset, where *CD* stands for cardinal numbers. This feature corresponds to the use of cardinal numbers as subject. This, again, provides a clear pointer to a specific mistake with the extraction process.

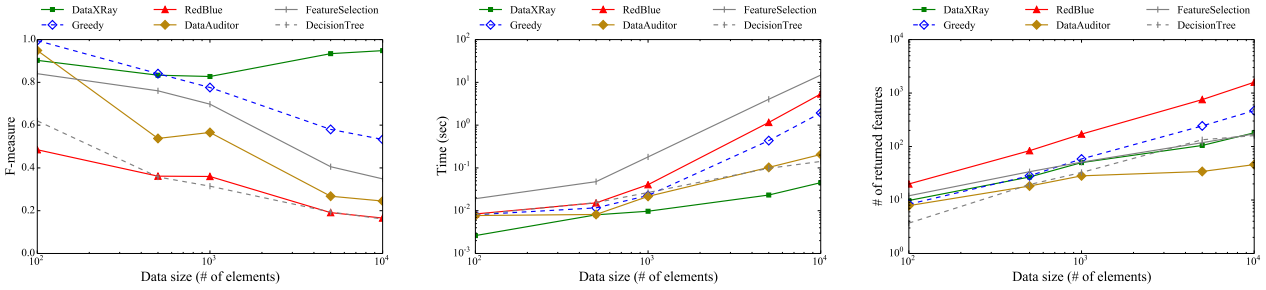
5.2 Scaling with synthetic data

We built a synthetic data generator to test our diagnostic framework across varied data sizes and error rates. We have three goals for these experiments: (1) evaluate how the different methods perform in terms of diagnostic accuracy across datasets of varied size and varied error rates, (2) evaluate how the different methods scale, and (3) evaluate the accuracy and scalability of DATAXRAY in a parallel setting across a large range of data sizes. All figures presented in this section display averages across 50 executions.

In the first round of our synthetic data experiments, we test all methods against datasets that range from 100 to 10,000 elements. In this experiment, each feature fails (becomes a cause of error) with probability 0.3, and the error rate of failed features is 0.95. We present the performance results in Figure 6. We note that DATAXRAY and DATAXRAY+GREEDY have almost identical performance in the synthetic data experiments (other than a negligible difference in execution time), so we omit the DATAXRAY+GREEDY line to make the plots less crowded. This is because our synthetic data generator avoids feature overlap at the same hierarchy levels, which makes the greedy refinement unnecessary. Therefore, from here on we do not include results for the greedy refinement.

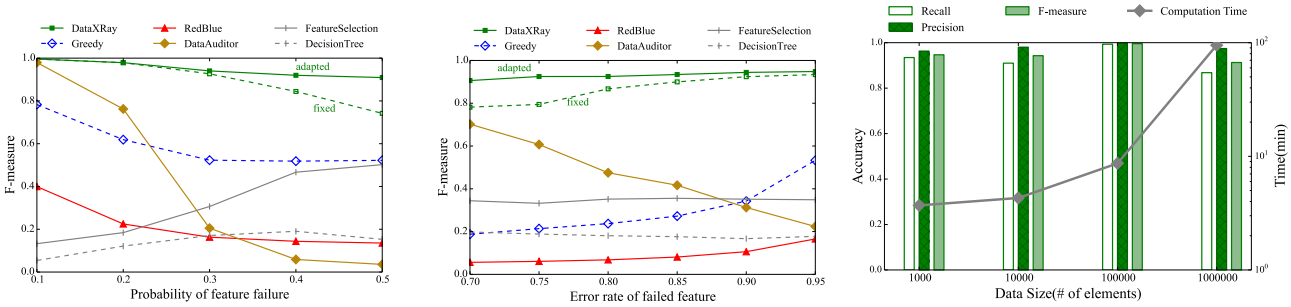
DATAXRAY is extremely effective across different data sizes, showing superior performance in both effectiveness and efficiency. As the size of the dataset increases, the F-measure of the competing methods steadily declines, falling below 0.6 for GREEDY and below 0.4 for the other methods at 10,000 elements. In contrast, our techniques maintain F-measure above 0.8 for all data sizes. DATAXRAY is also the fastest of the diagnostic methods across all data sizes, in several cases by multiple orders of magnitude (Figure 6b). Figure 6c evaluates the conciseness of the produced diagnoses. It is interesting that FEATURESELECTION derives diagnoses of very similar size to DATAXRAY, yet its F-measure is much lower, indicating that its objective is not suitable for the problem that we tackle in this work. DATAAUDITOR heavily favors features at higher levels of the hierarchy, which results in diagnoses with fewer features, but leads to low F-measure. We include additional results on the granularity of features chosen by different methods in the Appendix.

In our second round of experiments, we generate datasets of 10,000 elements but vary the probability that a feature is incorrect (Figure 7a) and the error rate among the elements of an incorrect feature (Figure 7b). As both these probabilities increase, they increase the overall error rate of the dataset, which leads us to consider ad-



(a) F-measure (b) Execution time (c) Number of returned features

Figure 6: Performance of different methods on synthetic data when we vary the size of the data. DATAXRAY maintains consistently good performance, while the effectiveness of the other methods drops dramatically as the size increases.



(a) F-measure (b) F-measure (c) Accuracy & execution time

Figure 7: Performance of different methods when we vary the parameters of our synthetic data generator. (a) and (b) show the robustness of DATAXRAY whereas (c) shows the scalability of our parallel implementation of DATAXRAY.

justments to the error pruning parameter (δ_{min}). Figures 7a and 7b display the performance for two versions of DATAXRAY: *fixed* uses fixed $\delta_{min} = 0.6$, while *adapted* adapts this parameter according to the overall error rate $\delta_{min} = \max(0.6, \epsilon_{\mathcal{E}})$.

Both versions of DATAXRAY maintain F-measure well above the other methods under this range of configurations, with DATAXRAY-adapted having a natural advantage. The performance of GREEDY decreases as the probability of feature failure increases (Figure 7a). The greedy approximation is prone to mistakenly rejecting large features from the diagnosis since they frequently contain `true` elements, which translates to higher feature weight. This occurrence is more common at high feature failure probability, where such features are more likely to be incorrect. On the other hand, FEATURESELECTION improves because the algorithm is better at selecting the larger incorrect features under the classification objective. When the error rate of an incorrect feature decreases, the performance of GREEDY drops (Figure 7b). This is also expected: with lower error rates, incorrect features include more correct elements, which leads the greedy approximation to a lot of mistakes. For DATAAUDITOR, increases in the overall error rate exacerbate its innate tendency to select features at higher hierarchy levels, leading to sharp drops in its performance. In contrast, the consistently high F-measure of DATAXRAY shows the robustness of our algorithm.

5.2.1 *Parallel evaluation*

We evaluated our framework with even larger data sizes using our MapReduce implementation of DATAXRAY (Section 4.3). Figure 7c presents results on our MapReduce DATAXRAY algorithm. We do not include data for the competing techniques, as we were not able to scale them beyond 10,000 elements, and we were not able to find comparable parallel implementations of these methods.[3]

We ran our parallel implementation of DATAXRAY on a Hadoop cluster with 15 slave nodes, ranging the data size from 1,000 to 1 million elements. Our results in Figure 7c show that the quality of our diagnoses does not degrade as the problem size grows larger. In addition, using a parallel framework allows us to derive diagnoses efficiently, even for large data sizes: our algorithm processes the 1 million elements in about 100 minutes and the execution time grows linearly with the data size. This execution time is very reasonable for this type of problem, as data diagnosis, similarly to data cleaning, is an offline process.

6. RELATED WORK

DATAXRAY targets the problem of data diagnosis: explaining where and how the errors happen in a data generative process. This is in contrast to traditional data cleaning [50], which focuses on identifying and repairing incorrect data. Identifying and correcting data errors is an important and well studied problem. Data management research has supplied a variety of tools to deal with errors originating from integrating data from multiple sources [1, 5, 47, 49], identifying data items that refer to the same entity [33, 37], resolving conflicts [22, 58, 59], and providing language extensions for cleaning [28]. The ultimate goal for all of these techniques is to identify which items in a dataset are correct, and which are incorrect. These techniques are complementary to our work; they can be used to label the truthfulness of elements in our diagnostic framework. Consequently, our work focuses on identifying mistakes in the process that produced the data, rather than finding errors in the dataset itself.

A major research thrust in the domain of data cleaning is to automatically generate repairs for the recovered errors [12]. Again, there is a large arsenal of tools in this category that use rules [7, 14] and functional dependencies [12, 26], while there is also work that focuses on generating repairs in an interactive fashion, using user

[3]To the best of our knowledge, existing parallel implementations of logistic regression target shared-memory architectures, so they are limited to shared-memory, multi-core systems [40].

feedback [51, 57]. DATAXRAY is a diagnostic tool, rather than a cure. It can offer insight on the likely causes of error, but it does not suggest specific repairs for these errors.

Data Auditor [30, 31] is a data quality exploration tool that uses rules and integrity constraints to construct *pattern tableaux*. These tableaux summarize the subsets of elements that fail to satisfy a constraint, with some flexibility to avoid over-fitting. The tableaux outputs resemble diagnoses with a list of features, and Data Auditor also uses a top down algorithm to derive them. However, there are several distinctions to DATAXRAY. First, users need to select the attributes and constraints that will be in the tableaux. Second, the objective function that it applies focuses only on the number of returned attributes (features), and the coverage of satisfying and non-satisfying elements is specified as confidence constraints. In our evaluation, we showed that Data Auditor has much lower F-measure than our methods. We also observed that its performance is extremely sensitive to the confidence constraint, but the optimal setting of this parameter was not consistent across data sizes and other experimental settings (Section 5).

As DATAXRAY traces errors in the processes that produce data, it has connections to the field of *data and workflow provenance*. Data provenance studies formalisms that express why a particular data item appears in a query result, or how that query result was produced in relation to input data [9, 11, 15, 32]. However, since we often do not know the details of each data generator (e.g., knowledge extractors), we cannot easily apply these approaches. In comparison, DATAXRAY describes how collections of data items were produced in relation to high-level characteristics of the data generative process. Roughly, features in our framework are a form of provenance annotations, but these are much simpler than the complex process steps and interactions that workflow provence [2, 19] typically captures and maintains. Work on interactive investigation of information extraction [18] uses provenance-based techniques and discusses a concept of diagnosis under a similar application setting. The focus of that work however, is on interactive exploration and repair tools, which is different from the scope of our work.

The work on database causality [41, 42, 44] refines the notions of provenance, and describes the dependencies that a query result has on the input data. Similar to DATAXRAY, causality offers a diagnostic method that identifies data items that are more likely to be causes of particular query outputs. This leads to a form of post-factum data cleaning, where errors in the output can be traced and corrected at their source [43]. There are four significant differences between causality and DATAXRAY. First, causal reasoning can only be applied to simple data generative processes, such as queries and boolean formulas, whereas DATAXRAY works with arbitrarily complex processes by relying on feature annotations. Second, existing algorithms for causality do not scale to the data sizes that we tackle with DATAXRAY. Third, existing work on causality in databases has focused on identifying fine-grained causes in the input data, which would correspond to individual elements in our setting, and no higher level features. Fourth, the premise of causality techniques relies on the assumption that errors, and generally observations that require explanation, are rare. This assumption does not hold in many practical settings that our diagnostic algorithms target.

Existing work on explanations is also limited in its applicability, as it is tied to simple queries or specific applications. The Scorpion system [56] finds predicates on the input data as explanations for a labeled set of outlier points in *an aggregate query over a single relation*. Roy and Suciu [52] extended explanations with a formal framework that handles complex SQL queries and database schemas involving multiple relations and functional dependencies. Predicate explanations are related to features, but these approaches are again limited to relational queries, rather than general processes. Finally, application-specific explanations study this problem within a particular domain: performance of MapReduce jobs [36], item rating [17,53], auditing and security [6,23]. Recent work [**?**] introduced sampling techniques (Flashlight and Laserlight) to derive explanation summaries. The objective of these methods is to find features that maximize the information gain. We evaluated Flashlight and Laserlight experimentally, but they faired very poorly when applied to our diagnostic problems (maximum F-measure below 0.1).

A natural step in the problem of data diagnosis is to use feature selection [46,54] to identify the features that can distinguish between the `true` and the `false` elements. There are different methods for performing feature selection, and logistic regression [8, 38] is one of the most popular. In our experiments, we used logistic regression with L_1 regularization to minimize the number of returned features. However, as our evaluation showed, it is not well suited for deriving good diagnoses. Moreover, machine learning methods such as logistic regression are hard to parallelize. Existing shared-memory implementations [40] offer significant speedups, but they cannot benefit from the MapReduce design that DATAXRAY uses. Efforts to implement feature selection in a MapReduce framework showed that this paradigm is not as effective as the shared-memory designs [39].

Other techniques like clustering [3, 4, 45] can be used to create groups of `false` elements as explanations of errors. However, most of the clustering work assumes non-overlapping clusters. This assumption makes these techniques ill-suited for our context as a `false` element may be caused by several reasons and it may be involved in multiple returned features.

Finally, the problem of finding an optimal diagnosis is related to different versions of the set cover problem [27]. Weighted set cover [16] seeks the set of features of minimum weight that covers all the `false` elements. By assigning feature weights using our cost model, we get an alternative algorithm for computing the optimal diagnoses. We use the greedy heuristic approximation for weighted set cover [13], but our evaluation showed that this algorithm is not as effective as DATAXRAY. Red-blue set cover [10, 48] is a variant of set cover; it seeks the set of features that cover all blue (`false`) elements, and tries to minimize the number of covered red (`true`) elements. Our experiments showed that the objective of red-blue set cover produces diagnoses of lower accuracy compared to DATAXRAY.

7. CONCLUSIONS

In this paper, we presented DATAXRAY, a large-scale, highly-parallelizable framework for error diagnosis. Diagnosis is a problem complementary to data cleaning. While traditional data cleaning focuses on identifying errors in a dataset, diagnosis focuses on tracing the errors in the systems that derive the data. We showed how to model the optimal diagnosis problem using feature hierarchies, and used Bayesian analysis to derive a cost model that implements intuitive properties for good diagnoses. Our experiments on real-world and synthetic datasets showed that our cost model is extremely effective at identifying causes of errors in data, and outperforms alternative approaches such as feature selection. By using the feature hierarchy effectively, DATAXRAY is also much faster than the other techniques, while our parallel MapReduce implementation allows us to scale to data sizes beyond the capabilities of the other methods.

Acknowledgements: We thank Evgeniy Gabrilovich, Ramanathan Guha, and Wei Zhang for inspiring discussions and use-case examples on Knowledge Vault. This work was partially supported by NSF CCF-1349784, IIS-1421322, and a Google faculty research award.

8. REFERENCES

[1] S. Abiteboul, S. Cluet, T. Milo, P. Mogilevsky, J. Siméon, and S. Zohar. Tools for data translation and integration. *IEEE Data Engineering Bulletin*, 22(1):3–8, 1999.

[2] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting lipstick on pig: Enabling database-style workflow provenance. *PVLDB*, 5(4):346–357, Dec. 2011.

[3] P. Arabie, J. D. Carroll, W. S. DeSarbo, and J. Wind. Overlapping clustering: A new method for product positioning. *Journal of Marketing Research*, 18:310–317, 1981.

[4] A. Banerjee, C. Krumpelman, J. Ghosh, S. Basu, and R. J. Mooney. Model-based overlapping clustering. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD, pages 532–537, New York, NY, USA, 2005. ACM.

[5] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, Dec. 1986.

[6] G. Bender, L. Kot, and J. Gehrke. Explainable security for relational databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 1411–1422, New York, NY, USA, 2014. ACM.

[7] G. Beskales, I. F. Ilyas, and L. Golab. Sampling the repairs of functional dependency violations under hard constraints. *PVLDB*, 3(1-2):197–207, Sept. 2010.

[8] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *International Conference on Machine Learning (ICML)*, Bellevue, Washington, June 2011.

[9] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[10] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 345–353, 2000.

[11] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[12] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469. IEEE Computer Society, 2013.

[13] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[14] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 315–326. VLDB Endowment, 2007.

[15] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems*, 25(2):179–227, 2000.

[16] M. Cygan, L. Kowalik, and M. Wykurz. Exponential-time approximation of weighted set cover. *Information Processing Letters*, 109(16):957–961, July 2009.

[17] M. Das, S. Amer-Yahia, G. Das, and C. Yu. Mri: Meaningful interpretations of collaborative ratings. *PVLDB*, 4(11):1063–1074, 2011.

[18] A. Das Sarma, A. Jain, and D. Srivastava. I4e: Interactive investigation of iterative information extraction. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 795–806, New York, NY, USA, 2010. ACM.

[19] S. B. Davidson and J. Freire. Provenance and scientific workflows: Challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 1345–1350, New York, NY, USA, 2008. ACM.

[20] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2014.

[21] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From data fusion to knowledge fusion. *PVLDB*, 2014.

[22] X. L. Dong and F. Naumann. Data fusion–resolving data conflicts for integration. *PVLDB*, 2009.

[23] D. Fabbri and K. LeFevre. Explanation-based auditing. *PVLDB*, 5(1):1–12, Sept. 2011.

[24] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *EMNLP*, 2011.

[25] W. Fan, F. Geerts, and X. Jia. A revival of integrity constraints for data cleaning. *Proc. VLDB Endow.*, 1(2):1522–1523, Aug. 2008.

[26] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems*, 33(2):6:1–6:48, June 2008.

[27] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, Apr. 1989.

[28] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: An extensible data cleaning tool. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD, page 590, New York, NY, USA, 2000. ACM.

[29] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[30] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376–390, Aug. 2008.

[31] L. Golab, H. J. Karloff, F. Korn, and D. Srivastava. Data auditor: Exploring data quality and semantics using pattern tableaux. *PVLDB*, 3(2):1641–1644, 2010.

[32] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[33] A. Gruenheid, X. L. Dong, and D. Srivastava. Incremental record linkage. *PVLDB*, 7(9):697–708, 2014.

[34] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. 2012.

[35] D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems*, 31(2):716–767, June 2006.

[36] N. Khoussainova, M. Balazinska, and D. Suciu. Perfxplain: debugging mapreduce job performance. *Proc. VLDB Endow.*, 5(7):598–609, Mar. 2012.

[37] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: Similarity measures and algorithms. In *Proceedings of the*

2006 ACM SIGMOD International Conference on Management of Data, SIGMOD '06, pages 802–803, New York, NY, USA, 2006. ACM.

[38] J. Liu, S. Ji, and J. Ye. *SLEP: Sparse Learning with Efficient Projections*. Arizona State University, 2009.

[39] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *PVLDB*, 5(8):716–727, Apr. 2012.

[40] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, Catalina Island, California, July 2010.

[41] A. Meliou, W. Gatterbauer, J. Halpern, C. Koch, K. F. Moore, and D. Suciu. Causality in databases. *IEEE Data Engineering Bulletin*, Sept. 2010.

[42] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.

[43] A. Meliou, W. Gatterbauer, S. Nath, and D. Suciu. Tracing data errors with view-conditioned causality. In *SIGMOD Conference*, pages 505–516, 2011.

[44] A. Meliou, S. Roy, and D. Suciu. Causality and explanations in databases. *PVLDB*, 7(13):1715–1716, 2014.

[45] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[46] A. Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *In ICML*, 2004.

[47] C. Parent and S. Spaccapietra. Issues and approaches of database integration. *Commununications of the ACM*, 41(5):166–178, 1998.

[48] D. Peleg. Approximation algorithms for the label-cover< sub> max</sub> and red-blue set cover problems. *Journal of Discrete Algorithms*, (1):55–64, March 2007.

[49] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, Dec. 2001.

[50] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[51] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 381–390, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[52] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 1579–1590, New York, NY, USA, 2014. ACM.

[53] S. Thirumuruganathan, M. Das, S. Desai, S. Amer-Yahia, G. Das, and C. Yu. Maprat: meaningful explanation, interactive exploration and geo-visualization of collaborative ratings. *PVLDB*, 5(12):1986–1989, Aug. 2012.

[54] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

[55] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, NAACL, pages 173–180, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[56] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, June 2013.

[57] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, Feb. 2011.

[58] X. Yin, J. Han, and P. S. Yu. Truth discovery with multiple conflicting information providers on the web. In *KDD*, pages 1048–1052, New York, NY, USA, 2007. ACM.

[59] B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han. A Bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.

APPENDIX

A. SUMMARY OF NOTATIONS

Notation	Description
$e(V, P)$	Element with truthfulness V and property vector P
$f(P, \mathbf{E})$	Feature with property vector P and set of elements $\mathbf{E}$
$\mathcal{E}$	Dataset of elements $\mathcal{E} = \{e_1, \ldots, e_n\}$
$\mathcal{F}$	Set of features $\mathcal{F} = \{f_1, \ldots, f_k\}$; candidate diagnosis
$\Pr(\mathcal{F}\|\mathcal{E})$	Causal likelihood: probability that $\mathcal{F}$ caused the errors in $\mathcal{E}$
$f.\mathbf{E}^-$	The `false` elements in f: $\{e \,\|\, (e \in f.\mathbf{E}) \wedge (\neg e.V)\}$
$f.\mathbf{E}^+$	The `true` elements in f: $\{e \,\|\, (e \in f.\mathbf{E}) \wedge e.V\}$
ϵ_i	The error rate of feature f_i: $\epsilon_i = \frac{\|f_i.\mathbf{E}^-\|}{\|f_i.\mathbf{E}\|}$
α	The *a priori* probability that a feature is a cause of error
SV_f	The structure vector of feature f
L_f	The level of feature f
$\mathcal{P}_i^{f_p}$	Partition i of the child features of f_p
Var_i^f	The variance of error rates of features in partition $\mathcal{P}_i^{f_p}$

Figure 8: Summary of notations used in the paper.

B. THEORETICAL RESULTS

THEOREM 15 (COMPLEXITY AND BOUNDS). *Algorithm 1 has complexity $O(|\mathcal{F}|)$, and provides an $O(n)$-approximation of the minimum cost diagnosis, where n is the number of elements in $\mathcal{E}$ and $\mathcal{F}$ is the set of features that can be derived from $\mathcal{E}$.*

PROOF. Let D be the diagnosis produced by Algorithm 1, and let D_{OPT} be the diagnosis of minimum cost that covers all the elements. Algorithm 1 compares the cost of a parent feature with that of its children, and decides to proceed if the cost decreases. That means that for every $F \in D$, $Ancestors(F) \cap D_{OPT} = \emptyset$: If F', ancestor of F, was in D_{OPT}, we could produce a diagnosis of lower cost than D_{OPT} by replacing F' with its children.

Therefore, D_{OPT} contains no ancestors of $F \in D$, but it could contain its descendants instead of F. Let n_F be the total number of elements under feature F, and let ϵ be the error rate of the feature. The cost of feature F is the sum of the three cost types in Definition 9:

$$C_F = \log \frac{1}{\alpha} + n_F \log \frac{1}{\epsilon^\epsilon (1-\epsilon)^{(1-\epsilon)}}$$

In the worst case, F can be replaced by a single descendant feature F'' with error rate 1. This means that the false and true costs of F'' are 0, and therefore, its overall cost is $C_{F''} = \log \frac{1}{\alpha}$. Since $\log \frac{1}{\epsilon^\epsilon (1-\epsilon)^{(1-\epsilon)}} \leq 1$, the cost of F is at most $O(n_F)$ worse than

the cost of the optimal descendant set of F. Adding these worst-case costs for all features in the diagnosis D, we get overall worst-case approximation bound $O(n)$, where n is the total number of elements.

Finally, Algorithm 1 accesses each feature at most once. Therefore, its time complexity is $O(|\mathcal{F}|)$. □

THEOREM 16 (TIGHTNESS). *The $O(n)$ approximation bound is tight.*

PROOF. Let F_0 be a feature that is the root of a subtree in the hierarchy. Let also F_0 have exactly two child nodes, F_1 and F_2, each containing n elements. Therefore, F_0 contains $2n$ elements. We assume that the error rates of features F_1 and F_2 are $\epsilon_1 = \epsilon_2 = \epsilon$. Therefore, the cost of F_0 is $C_0 = \log \frac{1}{\alpha} + 2n \log \frac{1}{\epsilon^\epsilon (1-\epsilon)^{(1-\epsilon)}}$, and the cost of F_1 and F_2 is $C_1 + C_2 = 2 \log \frac{1}{\alpha} + 2n \log \frac{1}{\epsilon^\epsilon (1-\epsilon)^{(1-\epsilon)}}$. Therefore, DATAXRAY will select F_0 instead of its children and will terminate the descent to that part of the hierarchy. However, the optimal diagnosis can be F_1' and F_2', descendants of F_1 and F_2 respectively, with total cost $2 \log \frac{1}{\alpha}$. This means that the cost C_0 is $O(n)$ worse than OPT. □

The worst-case analysis of the approximation bound only occurs when the errors are distributed uniformly across the children of a node, across all dimensions. If this is not true for some dimension, then the algorithm will descent into the corresponding partition to approach the optimal solution. It is in fact very unusual for this to happen in practice, which is why DATAXRAY performs much better than GREEDY in our experimental evaluation.

C. ADDITIONAL RESULTS

Figure 9 shows how the produced diagnoses compare with the ground truth at different granularities. For each level of the hierarchy, we depict the total incorrect features in a diagnosis (total false positive and false negative) for each method, normalized by the total number of features at that level. The plot presents an average of 50 executions over randomly generated datasets with 10,000 tuples and feature hierarchies of 5 levels.

DATAXRAY is the method closest to the ground truth, with only a few mistakes among features of average granularity (middle of the hierarchy). DATAAUDITOR tends to select more features at higher hierarchy levels, and this is where most of its mistakes are concentrated. REDBLUE and GREEDY make mistakes that are more evenly distributed across the hierarchy levels. In contrast, the classification techniques (FEATURESELECTION and DECISIONTREE) tend to make the most mistakes in the middle of the hierarchy, with almost 60% of the features at that level being incorrectly included to or excluded from the diagnosis.

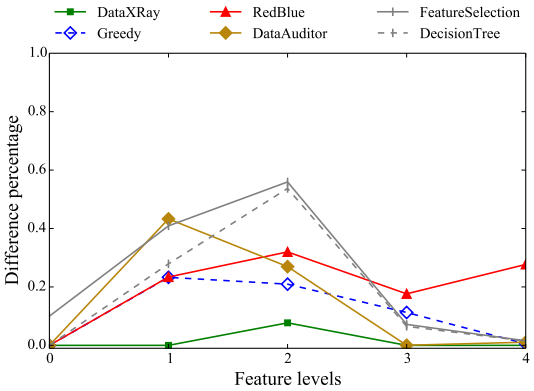


Figure 9: We evaluate how much the selected features at each hierarchy level deviate from the ground truth for each technique, over datasets of 10,000 elements. Level 0 is the root of the hierarchy, and level 4 contains the leaves (individual elements). DATAXRAY has the smallest difference from the ground truth.