Tracing Data Errors with View-Conditioned Causality*

Alexandra Meliou† Wolfgang Gatterbauer† Suman Nath§ Dan Suciu†

†University of Washington,
Seattle, WA, USA
{ameli,gatter,suciu}@cs.washington.edu

§Microsoft Research,
Redmond, WA, USA
Suman.Nath@microsoft.com

ABSTRACT

A surprising query result is often an indication of errors in the query or the underlying data. Recent work suggests using causal reasoning to find explanations for the surprising result. In practice, however, one often has *multiple queries* and/or *multiple answers*, some of which may be considered correct and others unexpected. In this paper, we focus on determining the causes of a set of unexpected results, possibly conditioned on some prior knowledge of the correctness of another set of results. We call this problem *View-Conditioned Causality.* We adapt the definitions of causality and responsibility for the case of multiple answers/views and provide a non-trivial algorithm that reduces the problem of finding causes and their responsibility to a satisfiability problem that can be solved with existing tools. We evaluate both the accuracy and effectiveness of our approach on a real dataset of user-generated mobile device tracking data, and demonstrate that it can identify causes of error more effectively than static Boolean influence and alternative notions of causality.

Categories and Subject Descriptors. E.0 Data General
General Terms. Algorithms
Keywords. causality, view-conditioning, error tracing

1. INTRODUCTION

Data transformations from a source to a target dataset are ubiquitous today and can be found in data integration [26], data exchange [25, 3], and ETL tools [31]. Users often detect *errors in the target data.* For example, a user may detect that an item in a target data instance is incorrect: the tuple should not be there, or some of its attribute values are erroneous; she would like to find out which of the many input tuples that contributed to the incorrect output is faulty. It is critical that the error be *traced and corrected in the source data*, because once an error is identified, one can prevent it from propagating to multiple items in the target data. This can be viewed as a form of "post-factum" data cleaning: while in standard data cleaning one corrects errors before the data is transformed and integrated, in our setting the errors are detected only *after the data has been transformed.*

*This work was partially funded by NSF IIS-0911036, IIS-0915054, and IIS-0713576.

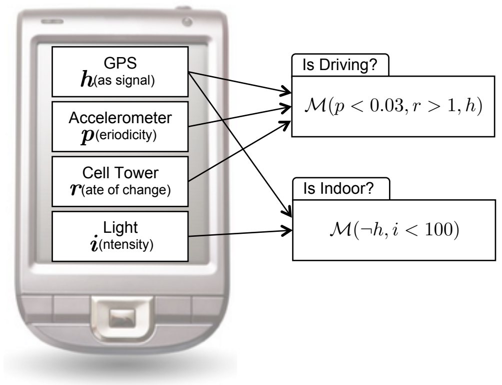


Figure 1: Two simple classifiers based on sensory input from a cell phone. The inputs to the classifiers are either Boolean variables (e.g. h), or scalar values to which an appropriate threshold is applied (e.g. i). Symbol $\mathcal{M}$ denotes a strict majority function.

In this paper, we show how to extend the concept of causality [28] to *trace errors in a source dataset that caused incorrect results in a target dataset.* We showcase our motivation through a practical example.

EXAMPLE 1 (RECOMMENDATION SYSTEM). *Consider a new generation smart phone. It has multiple sensors (e.g. GPS, accelerometer, light, and cell tower signal). Based on these sensors, a set of classifiers can predict the owner's current activities (e.g. walking, driving, working, being with family, or being in a business meeting). Using the knowledge of the user's current activities allows the system to serve the user with useful targeted recommendations. For example, if the user is away from their car around lunch time, the application will suggest restaurants within walking distance.*

This is an example of data transformation: the source data (input) are the sensors, the target data (output) are the activities. Inaccuracies in the sensor data are a common occurrence: sensors have innate imprecisions (e.g., the GPS may miscalculate the current location), or some sensors may become inhibited (e.g., if the user places the cell-phone in the glove compartment while driving, then the light sensor's reading is incorrect). As a consequence, an inferred activity may be wrong. The application can often detect such errors from user feedback or based on the user's subsequent actions and reactions to the provided recommendations. But *the main challenge is to actually identify the responsible sensor(s). For example, in Fig. 1 the "is-driving" activity depends on 3 sensors: if "is-driving" is wrong, which of the three sensors is erroneous? With this knowledge, the system could inhibit the reading from that sensor and therefore improve the other classifiers.*

The first step towards tracing errors in the source data is to compute the lineage of each item in the target data. Data lineage, or provenance, has been extensively studied recently [10, 16, 5]. But as Example 1 shows, examining the lineage of an erroneous result is not sufficient to identify the cause of the problem: all three sensor inputs are part of the lineage, and without further context, it is not possible to determine which specific sensor is erroneous.

In this work, we identify erroneous inputs by building upon a long history of work on *causality theory.* Albeit related to lineage, causality is a more refined notion. The lineage of an erroneous output reflects causal dependencies between the input and output data, and causal reasoning can use this information to return the possible causes ranked by their *degree of responsibility.* Causality has been studied extensively in philosophy, AI, and cognitive science. We base our discussion on the established definitions of *actual causes* and *responsibility* first developed by Halpern, Pearl and Chockler [11, 17], and our previous work that adapted these definitions to a database context, targeting query result explanations [28].

In error tracing, however, we face a richer context than in traditional causality theory. In addition to the erroneous output item, we also have access to several output items that we know are correct. In fact, there may even be more than one erroneous output item. Accounting for this richer context leads to much better error tracing.

Example 2 (Recommendation System - continued). *A modern smart phone typically has 10-12 sensors, and could have up to 20-30 classifiers, predicting a variety of activities. Based on the user's response to recommendations, the system can typically derive that multiple predictions are correct or incorrect. Suppose, for example, that the system knows that 3 predictions are wrong and 2 are correct. Now the system has more information than just knowing that one classifier is wrong. The challenge is to find the most likely input sensor that could have caused the 3 faulty predictions, while at the same time allowing the 2 correct predictions. Thus,* conditioned *on the two correct outputs, we seek the most responsible sensor(s) that explain the faulty classifiers.*

For example, assume the two simple classifiers of Fig. 1: if both are incorrect, then intuitively, the GPS becomes the most likely culprit; if the "is-driving" classification is wrong and "is-indoor" is correct, then we know that the GPS input has to be correct.

In this paper, we propose *View-Conditioned Causality*, a novel concept that allows us to trace errors in transformed target data, such as query results or views, back to the input base data. Our main contributions are as follows:

(1) We propose a new definition of view-conditioned causality and its associated notion of responsibility. In order to use causality for error tracing, we refine existing notions of causality/responsibility by conditioning on all knowledge of correct or incorrect results.

(2) Computing causality and responsibility is known to be hard in general [14]. Previously, we have studied the complexity of causality and responsibility for conjunctive queries [28]. Despite some tractable cases, computing the responsibility remains hard, in general. To date, no practical algorithms exist to compute causes or responsibilities for all cases. In order to compute causality and responsibility in practice, we propose a novel algorithm for translating the view-conditioned causality and responsibility problems into a SAT and a weighted MAX-SAT problem respectively, for which very effective tools exist [2, 4]. In the case of causality, our algorithm produces a Boolean expression Φ_{SAT} that is satisfiable iff a given input variable is a view-conditioned cause. In the case of responsibility, our algorithm also produces, in addition to the "hard" constraint Φ_{SAT}, several "soft" clauses: the larger the number of soft clauses that cannot be satisfied, the lower the responsibility.

(3) The formula Φ_{SAT} needs to be converted to CNF in order to be processed by general (Max)SAT solvers, and this has the potential of an exponential blow-up. Our third contribution describes an optimized conversion to CNF that results in exponentially smaller CNF expressions.

(4) Finally, we perform an experimental validation of our techniques on real data from an actual application analogous to Example 1. Our experiments verify the quality of our approach in identifying erroneous inputs, and evaluate its performance. We show that view-conditioned causality retains an average precision very close to 1, outperforming other techniques such as counterfactual causality and Boolean influence, by 60% and 75% respectively at certain instances. Our optimized algorithm for the CNF conversion is shown to produce SAT problems orders of magnitude smaller than the naive approach, while the SAT solvers were able to process even the larger generated problem instances in a few seconds. We further demonstrate interesting insights on how responsibility of features can function as a meaningful quantifier of classifier quality and robustness: classifiers that use features with low responsibility can be easily corrected by eliminating faulty inputs, whereas classifiers whose features are all of high responsibility (i.e. they are all likely sources of failure), cannot achieve good correction rates.

The structure of this paper follows its main contributions:

- We propose view-conditioned causality and responsibility as a solution for tracing errors from views and transformations to source data (Sect. 3).
- We demonstrate a non-trivial algorithm that reduces the problems of computing causes and responsibility to the satisfiability of Boolean expressions which can be solved with existing tools (Sect. 4).
- We propose an optimized conversion algorithm of the resulting Boolean expressions into CNF, in order to be fed into a SAT or MaxSAT solver (Sect. 5).
- We illustrate the effectiveness of using our techniques for tracking errors in a real world classifier-based recommendation system (Sect. 6).

2. BACKGROUND AND RELATED WORK

Causality is an active area of research, mostly in AI and philosophy. One of the basic notions in causality theory is *counterfactual causality* [29, 27]. An input variable is counterfactual if a change in its value reverses the value of the output variable. Counterfactuals are a very intuitive notion, but very limited in their applicability as they don't support disjunctive causes: If Z occurs when X_1 or X_2 occurs, neither of X_1 or X_2 are counterfactual causes for Z.

Halpern and Pearl [17] extended counterfactuals by introducing *actual causes* which rely upon a graph structure called a *causal network*, and added the notion of *permissive contingencies*. The causal network is defined by *structural equations*, which describe how variables relate to each other; for example $Z = X_1 \vee X_2$ is a structural equation linking inputs X_1 and X_2 to output Z. Contingencies are based on the notion of *intervention*: what becomes counterfactual if part of the input changes. For example, if X_1 and X_2 are both `true`, neither of them is counterfactual for Y. However, if the input changes in a certain way, for instance X_2 is `false`, then X_1 becomes counterfactual. One says that X_1 *is an actual cause of Z under the contingency* $X_2 =$ `false`. Such contingencies can be viewed as alternative possible worlds or variable assignments. Roughly, a variable X is an actual cause if there exists a contingency (a value assignment for the other variables) that makes X counterfactual.

The notion of *responsibility* was first defined in [11] as a measure of the degree of causality: the responsibility is inverse proportional to the size of the smallest contingency set. This is a key notion for our setting, because responsibility allows us to rank causes. Ranking is critical in complex queries with large lineages, and in the case of error tracing it serves as a comparison metric of the likelihood of the various error sources. However, determining causes and responsibility was shown to be NP-hard in general [14].

Very recently, we extended the notions of causality and responsibility to database queries [28]. Here the input variables are tuples in the input database, the output variables are the answer tuples returned by the query, and the causal network is given by the *lineage expression*. An input tuple is a *counterfactual cause* for an output tuple if its removal from the input causes the output to disappear. An input tuple is an *actual cause* for an output tuple if there exists a set of tuples in the input (the contingency) such that, after removing them, the input tuple becomes a counterfactual cause. The complexity of causality and responsibility were analyzed as a function of the query (query complexity). These results are unrelated to our work, since we analyze directly the lineage expression and do not consider query complexity. Neither the general work on causality, nor its adaptation to database queries have studied the problem of tracing errors from output data to input data.

Our work relates to prior work on data provenance, in the sense that our input is a provenance expression. There, approaches are mainly classified into three categories: how-, why-, and where- provenance [6, 10, 13, 16]. Also, recent work has focused on the problem of explaining missing query results, i.e. why a certain tuple does not appear as an answer. The work in this field is divided into data-focused [22, 20, 19] where explanations are given as base tuples, and query-focused [8, 33] where explanations are based on query predicates. The ability to *rank causes* clearly distinguishes causality from the provenance and missing answers work, and makes it more appropriate for tracking errors. In independent work, Cheney [9] also discusses the analogy between provenance and causality, and describes defining causal semantics for provenance graphs.

An interesting, and related work is the view side-effects problem on tuple deletion and annotation propagation [7]. This problem involves finding a subset of tuples in the database whose deletion will eliminate a given tuple t from the view, with the minimum number of side-effects (other tuple eliminations) in the view. An analogous definition of the problem exist for tuple insertion and updates, and all are shown to be hard, in general. Very recent work [12] studies the "side-effect free" version of this problem, where zero side-effects are enforced, and the aim is to minimize the subset of tuples selected from the database. Here, the same hardness results hold as in the general version of the problem. Note that these problems differ from computing responsibility, where we need to find the minimum set of input tuples that make a given tuple counterfactual. Also, the view side-effects problem is in PTIME for queries that do not contain joins together with projection or union. In contrast, we showed the hardness criterion for computing responsibilities to be entirely different [28], and even queries that only contain joins can be NP-hard.

3. VIEW CONDITIONING

Formal setup. We use capital letters for variables (e.g. X), small letters for value assignments (e.g. x), and bold letters for sets of variables or assignments (e.g. $\boldsymbol{X}$ or $\boldsymbol{x}$). We denote Boolean values interchangeably with `true` and `false` or `T` and `F`, respectively. We consider a set of n *input variables* (also called parameters) $\boldsymbol{X} = \{X_1, \ldots, X_n\}$, and a set of observed *output variables* $\boldsymbol{Z} = \{Z_1, \ldots, Z_m\}$. Each input variable X_i takes values from a discrete or continuous domain, e.g. reals, integers, Booleans. Each output variable Z_j is Boolean. We denote $\boldsymbol{\Phi} = \{\Phi_1, \ldots, \Phi_m\}$ a transformation from the input variables to the output variables. Each Φ_j is called a *lineage expression*, and is a Boolean expression over the input variables, defined as follows. Call a *threshold predicate* a predicate of the form X_i `op` t, where X_i is an input variable, `op` is one of $<, \leq, =, \neq \geq, >$, and t is a constant threshold value. Then Φ_j is any Boolean expression over threshold predicates. Thus, the transformation $\boldsymbol{\Phi}$ takes an input vector $\boldsymbol{x}$ of values, and computes the output vector $\boldsymbol{z}$, where each $z_j = \Phi_j(\boldsymbol{x})$. We write $\boldsymbol{x} \models \boldsymbol{z}$, to denote that $\boldsymbol{z}$ are the output values of the view transformations given input values $\boldsymbol{x}$. In addition to the input and output vectors, we further assume to know the correct values for the output $\boldsymbol{Z}$ and denote this *ground truth* as $\hat{\boldsymbol{z}} = \{\hat{z}_1, \ldots, \hat{z}_m\}$. If $z_i = \hat{z}_i$ then variable Z_i is correct, otherwise it is incorrect.

For a simple illustration, consider a simple classifier that computes $Z = (X_1 > 10) \wedge (X_2 < 3)$. Given two input values x_1, x_2 the classifier returns the output `T` or `F`, in an obvious way. Classifier systems are more complex types of transformations. For a more traditional transformation example, consider a select-project-join query. Each input tuple corresponds to a Boolean variable X_i, and each output tuple is a Boolean variable Z_j. The formula Φ_j is the standard lineage expression giving the dependency of Z_j as a function of the input tuples [16].

Note that our discussion will focus on classifier systems, but our results can be invariably used for other transformation types like conjunctive queries as well. Even though data sizes are much larger in query scenarios than sensor based applications, the lineage of each output tuple is actually bound by the query size. Moreover, we only need to keep as relevant variables those that participate in the lineage of erroneous outputs, thus greatly reducing the problem size. Finally, the SAT solver component is not prohibitive to this end, as modern SAT solvers can solve problems having millions of variables and clauses in less than 10 minutes of run time (cf. [1]).

3.1 The Problem: Tracing Data Errors

By comparing the output values $\boldsymbol{z}$ to the ground truth $\hat{\boldsymbol{z}}$, we can detect errors in the output. This means that there is an error in the input data $\boldsymbol{X}$, but we don't know which one, and only know which output is erroneous. Thus, while we may learn the ground truth for the output values, we don't know the ground truth for the input variables. For example, in Example 1 it is not possible to know whether a sensor is inhibited by looking at its measurements, but it is quite possible to determine that a classifier was incorrect, by examining the user's actions. Our goal is to trace the output errors back to the input data, and this justifies the following problem:

Error tracing problem: *Given input values $\boldsymbol{x}$ for the input variables $\boldsymbol{X}$, a set of transformations $\boldsymbol{\Phi}$ computing the values $\boldsymbol{z}$ of the output variables $\boldsymbol{Z}$, and given a ground truth $\hat{\boldsymbol{z}}$ for the output variables, detect the sources of error in the input data.*

Clearly, if $\boldsymbol{z} = \hat{\boldsymbol{z}}$ then there are no errors. We will assume that $\boldsymbol{z} \neq \hat{\boldsymbol{z}}$ for the rest of the paper. In our problem setting, the ground truth includes values for all output variables $\boldsymbol{Z}$. In practice that is often not the case: we know the ground truth for some, but not all output variables. In that case we simply restrict the output variables to those for which the ground truth is known. Thus, in the rest of this paper we will assume that the ground truth is known for all the output variables $\boldsymbol{Z}$.

3.2 Our Approach: View-Conditioned Causality

We define here counterfactual causes, and actual causes for a *set* of output values, which we call *view-conditioned causality.* Then, we explain how this can be applied to tracing errors. As a first attempt we may try to say that an input X_i is a *view-conditioned counterfactual cause* (or VCC cause, in short), if a change in the value x_i "corrects" the output $\boldsymbol{z}$ to the ground truth output $\hat{\boldsymbol{z}}$; we say in this case that X_i is a VCC cause of $\boldsymbol{z}|\hat{\boldsymbol{z}}$, and read that "$\boldsymbol{z}$ is conditioned on $\hat{\boldsymbol{z}}$". However, this is a tall order. The problem is that we are requiring that X_i, alone, can correct simultaneously *all* erroneous output values, while keeping unchanged *all* correct output values. Instead, our definition considers an entire set of input variables to be a counterfactual cause:

DEFINITION 3 (VCC CAUSE). *Consider input values $\boldsymbol{x}$ for the input variables $\boldsymbol{X}$, a transformation $\boldsymbol{\Phi}$ computing the output values $\boldsymbol{z}$, and a ground truth $\hat{\boldsymbol{z}}$. A set $\boldsymbol{X}_c \subseteq \boldsymbol{X}$ is a* view-conditioned counterfactual cause *of $\boldsymbol{z}|\hat{\boldsymbol{z}}$, if there exist values $\hat{x}_i$ for each $X_i \in \boldsymbol{X}_c$, such that the following two conditions hold:*

- $\boldsymbol{x} \models \boldsymbol{z}$ *and*

$$\{\{X_i = x_i \mid X_i \notin \boldsymbol{X}_c\} \cup \{X_i = \hat{x}_i \mid X_i \in (\boldsymbol{X} \backslash \boldsymbol{X}_c)\}\} \models \hat{\boldsymbol{z}}$$

- $\boldsymbol{X}_c$ *is minimal, i.e. no subset of $\boldsymbol{X}_c$ is a VCC cause.*

Intuitively, a VCC cause is a minimal set of input variables for which there exists a changed assignment that results in output $\hat{\boldsymbol{z}}$.

EXAMPLE 4 (VCC CAUSES). *Consider the views $\boldsymbol{Z}$ given in Fig. 2b over the inputs $\boldsymbol{X}$, where X_1 and X_3 take values*

$\boldsymbol{X}$	$\boldsymbol{x}$
X_1	5
X_2	T
X_3	2

(a)

$\boldsymbol{Z}$	$\boldsymbol{\Phi}$	$\boldsymbol{z}$
Z_1	$(X_1 < 10) \wedge X_2$	T
Z_2	$(X_3 > 0) \wedge X_2$	T
Z_3	$(X_3 > 3)$	F

(b)

Figure 2: (a): Value assignments for inputs $\boldsymbol{X}$. (b): The values of outputs $\boldsymbol{Z}$ are based on simple transformations (also called lineages) $\boldsymbol{\Phi}$ of the inputs $\boldsymbol{X}$.

from the integer domain, while X_2 is Boolean. Based on the input values from Fig. 2a, the observed output values are $\boldsymbol{z} = \{z_1, z_2, z_3\} = \{\mathtt{T}, \mathtt{T}, \mathtt{F}\}$. Assume that the ground truth is $\hat{\boldsymbol{z}} = \{\underline{\mathtt{F}}, \mathtt{T}, \underline{\mathtt{T}}\}$, where the erroneous results are underlined. Suppose first that we restrict the output variable to Z_1 (thus, ignore Z_2, Z_3). Both X_1 and X_2 are counterfactual causes of Z_1: changing the value of X_1 (e.g. $X_1 = 12$), or the value of X_2 from T *to* F*, would make Z_1* F*. However, if we take into account all three output variables, then X_2 is* not *a VCC cause of $\boldsymbol{z}|\hat{\boldsymbol{z}}$: if X_2 is switched to* F*, Z_1 would be "corrected" but Z_2 would also become* F*, which would contradict the ground truth $\hat{\boldsymbol{z}}$. In this example, there is a unique VCC cause of $\boldsymbol{z}|\hat{\boldsymbol{z}}$, which is the set of inputs $\{X_1, X_3\}$: switching their values to 12 and 4, respectively, sets all views to the desired output $\hat{\boldsymbol{z}}$.*

Actual causes in view-conditioned causality are defined in a similar manner based on contingency sets.

DEFINITION 5 (VC CAUSE). *Consider input values $\boldsymbol{x}$ for the input variables $\boldsymbol{X}$, a transformation $\boldsymbol{\Phi}$ computing the output values $\boldsymbol{z}$, and a ground truth $\hat{\boldsymbol{z}}$. A variable $X_i \in \boldsymbol{X}$ is a* view-conditioned cause *(VC cause) of $\boldsymbol{z}|\hat{\boldsymbol{z}}$, if there exists a set $\Gamma \subset \boldsymbol{X}$ called the* contingency *of X_i, such that $\{X_i\} \cup \Gamma$ is a VCC cause of $\boldsymbol{z}|\hat{\boldsymbol{z}}$.*

Note that every variable X_i in a VCC cause $\boldsymbol{X}_c$ of $\boldsymbol{z}|\hat{\boldsymbol{z}}$ is an VC cause with contingency $\boldsymbol{X}_c \setminus \{X_i\}$. Also, every VCC cause is a VC cause with an empty contingency set. Therefore, in Example 4, X_1 is a VC cause of $\boldsymbol{z}|\hat{\boldsymbol{z}}$ with contingency X_3, and X_3 is a VC cause with contingency X_1. Hence, even though in our example only sets of input variables are VCC causes, there are also individual input values which are VC causes.

Finally, we define the notion of responsibility, which measures the degree of causality.

DEFINITION 6 (RESPONSIBILITY). *Let $\boldsymbol{X}$ be a set of inputs to a set of views $\boldsymbol{Z}$, and assume $\boldsymbol{z}$ and $\hat{\boldsymbol{z}}$ to be the actual and ground truth values of those views, respectively. If $X_i \subseteq \boldsymbol{X}$ is a VC cause of $\boldsymbol{z}|\hat{\boldsymbol{z}}$, then its responsibility is defined as*

$$\rho_{X_i} = \frac{1}{1 + \min_\Gamma |\Gamma|}$$

where Γ is a contingency for X_i.

The responsibility is a function of the minimal number of input variables that need to be modified together with X_i in order to achieve the output $\hat{\boldsymbol{z}}$ for the views. If X_i is counterfactual, then its contingency set is empty, and therefore $\rho_{X_i} = 1$. If X_i is not a cause, by convention $\rho_{X_i} = 0$.

Our approach to the error tracing problem: *Given $\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{\Phi}, \hat{\boldsymbol{z}}$, we will compute all causes X_i, and rank them by their responsibility ρ_{X_i}. This ranking is a good indicator for tracing the errors in the input data.*

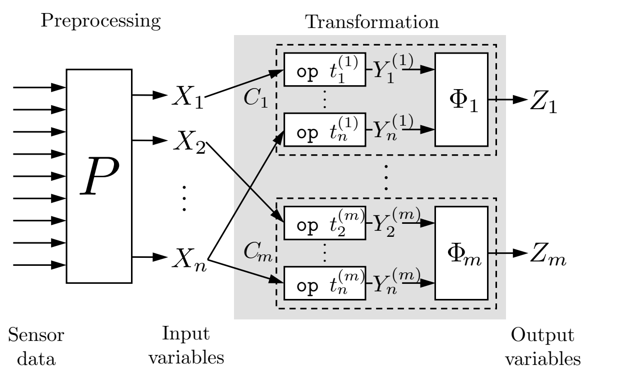


Figure 3: Structure of a classifier system.

We end this section by comparing our definition of causality and responsibility to prior work [17, 28]. In the prior definition, there is only one output variable, and this allows the definition of a counterfactual cause to be restricted to a single input variable. As we have seen, in error tracing applications we need to consider multiple output variables (some correct and some incorrect), and here the simple definition of counterfactual cause no longer works (as we saw in Example 4). Our new Def. 3 is an important extension to sets of counterfactual causes. On the other hand, this extension allows us to simplify (somewhat) the definition of an actual cause, in that the contingency set Γ is simply part of a counterfactual cause.

EXAMPLE 7. *Continuing our running example, we show it can be modeled in our formalism. Our application is a classification system with several sensor signals, and several classifiers determining user activities and status based on the sensor data. Sensor measurements can be in general vectors of values, like a time sequence of a signal. Before this data can be useful for the classification, it often undergoes some initial mathematical transformation to compute several useful metrics of the input signals (e.g. frequency, signal strength etc). We are not interested in the raw signal, but rather these computed metrics that serve as inputs to the classifiers. These represent our input* $\boldsymbol{x} = \{x_1, \ldots, x_n\}$. *There are* m *classifiers* $\{C_1, \ldots, C_m\}$*, producing a set of outputs* $\boldsymbol{Z} = \{Z_1, \ldots, Z_m\}$. *As an example,* $\boldsymbol{Z}$ *may represent a classification of the user's current activities (walking, driving, in a meeting).*

An abstraction of the architecture of the system is shown in Fig. 3. The classifier inputs $\boldsymbol{x}$ *are continuous values, or Boolean values. Each classifier* C_j *uses a subset of the transformed values* $\boldsymbol{x}_j \subseteq \boldsymbol{x}$ *to compute a classification result* Z_j. *Within each classifier* C_j*, each variable* $x_i \in \boldsymbol{x}_j$ *is compared against one or more predetermined thresholds, resulting in a Boolean variable* $Y_i^{(j)}$. *These thresholds are usually selected using machine learning techniques over some training data. In general,* $Y_i^{(j)} = (x_i \text{ op } t_i)$*, where op is one of* $\{<, >, \leq, \geq\}$*, and* t_i *is a threshold value. The final output* Z_j *of the classifier is determined by a Boolean function* $\Phi_j(\boldsymbol{Y}_j)$.

4. REDUCTION TO SAT AND MAXSAT

Computing the causality and responsibility are known to be NP-hard [14], and the same holds for view-conditioned-causality. Our approach for tracing errors to the data source is to reduce the causality/responsibility problem to the SAT problem, and to partial weighted MaxSAT problem respectively. This is an intensively studied research area (an extensive survey is in [4]) and there exist several highly optimized tools both for SAT and weighted MaxSAT. In this section we describe an algorithm for reducing the causality/responsibility problem to the SAT/MaxSAT problem. In the next section we show how to convert the resulting Boolean expressions into CNF, so we can feed them directly into some of these tools.

The algorithm consists of the following steps

1. Map from input variables with continuous domain to Boolean variables called *partitioned* variables.
2. Construct the constraint expression Ψ that captures the correlations between the partitioned variables.
3. For each input X_i, construct a Boolean expression Φ_{SAT} that is satisfiable iff X_i is a VC cause. Φ_{SAT} is a hard constraint.
4. Construct "soft" constraints to account for the contingency set: the more soft constraints are violated while satisfying Φ_{SAT} the larger the contingency set.

4.1 Mapping to Boolean Variables

Recall that while all the outputs $\boldsymbol{Z}$ are Boolean variables, some of the inputs $\boldsymbol{X}$ can be numerical values, and the Boolean expressions $\boldsymbol{\Phi}$ are over threshold predicates over these inputs, e.g., $\Phi_3 = (X_1 < 2.5) \vee (X_6 > 20)$. Since we need to map all inputs to Boolean variables, a first attempt is to create one Boolean variable for each threshold predicate, call it a *threshold variable*. An example threshold variable is $Y_1^{(3)} \equiv (X_1 < 2.5)$, where the subscript 1 refers to the input variable X_1, and the superscript (3) refers to the classifier Φ_3. But threshold variables are not independent, and this prevents us from using them in our translation.

EXAMPLE 8 (BOOLEAN MAPPING). *Assume 3 classifications over 6 inputs:*

$$\begin{aligned}
\Phi_1 &= \big((X_1 > 10) \wedge (X_2 < 7)\big) \vee \big((X_5 > 10) \wedge (X_3 > 20)\big) \\
&= \big(Y_1^{(1)} \wedge Y_2^{(1)}\big) \vee \big(Y_5^{(1)} \wedge Y_3^{(1)}\big) \\
\Phi_2 &= \big((X_1 < 4) \wedge (X_4 < 5)\big) \\
&= \big(Y_1^{(2)} \wedge Y_4^{(2)}\big) \\
\Phi_3 &= (X_1 < 2.5) \vee (X_6 > 20) \\
&= Y_1^{(3)} \vee Y_6^{(3)}
\end{aligned}$$

The three threshold variables $Y_1^{(1)}$, $Y_1^{(2)}$ *and* $Y_1^{(3)}$ *for the input* X_1 *are dependent in different ways:* $Y_1^{(1)}$ *cannot be true together with* $Y_1^{(2)}$ *or* $Y_1^{(3)}$*, whereas there exist values of* X_1 *for which* $Y_1^{(2)}$ *and* $Y_1^{(3)}$ *can both be true.*

Therefore, we take a different approach. For a given variable X_i, let $\{t_1, \ldots, t_{T_i}\}$ be the set of all thresholds that appear in threshold predicates associated with variable X_i across all transformations. We define a set of $2T_i+1$ Boolean *partition variables* $\boldsymbol{X}_{[i]} = \{X_{[i,1]}, \ldots, X_{[i,2T_i+1]}\}$ as follows:

$$\begin{aligned}
X_{[i,1]} &\equiv (X_i < t_1) & & \\
X_{[i,2j]} &\equiv (X_i = t_j) & & 1 \leq j \leq T_i \\
X_{[i,2j+1]} &\equiv (t_j < X_i < t_{j+1}) & & 1 \leq j < T_i \\
X_{[i,2T_i+1]} &\equiv (X_i > t_{T_i}) & &
\end{aligned}$$

Denote $\boldsymbol{X}_{[]}$ the set of all partition variables, $\boldsymbol{X}_{[]} = \bigcup_i \boldsymbol{X}_{[i]}$. Then, we rewrite each threshold predicate as a disjunction of these variables, and we rewrite each Boolean expression Φ_j in terms of the partitioned variables, denoting $\Phi_j^{[]}$ the rewritten expression; when it is clear from the context, we drop the superscript [] and use Φ_j also for the translated expression. Note that the use of partition variables only linearly increases the number of variables with the number of thresholds per variable, with two new variables created for every threshold.

Example 9 (Example 8 continued). *The partition variables of X_1 are given as:*

$$X_{[1,1]} \equiv (X_1 < 2.5) \qquad X_{[1,5]} \equiv (4 < X_1 < 10)$$
$$X_{[1,2]} \equiv (X_1 = 2.5) \qquad X_{[1,6]} \equiv (X_1 = 10)$$
$$X_{[1,3]} \equiv (2.5 < X_1 < 4) \qquad X_{[1,7]} \equiv (X_1 > 10)$$
$$X_{[1,4]} \equiv (X_1 = 4)$$

The partition variables divide the domain of X_1 into non-overlapping ranges, and therefore exactly one of them is true for any given value x_1. The rewritten threshold predicates are:

$$Y_1^{(1)} \equiv X_{[1,7]}$$
$$Y_1^{(2)} \equiv X_{[1,1]} \vee X_{[1,2]} \vee X_{[1,3]}$$
$$Y_1^{(3)} \equiv X_{[1,1]}$$

Thus, each $\Phi_j^{[]}$ now depends on the new partition variables.

Every value assignment $\boldsymbol{x}$ of $\boldsymbol{X}$ defines a unique Boolean assignment $\theta_{\boldsymbol{x}}$ of $\boldsymbol{X}_{[]}$, such that $X_{[i,j]}$ is true in $\theta_{\boldsymbol{x}}$ iff the value $X_i = x_i$ satisfies the partition predicate corresponding to the variable $X_{[i,j]}$. Then, one can check that, for every j, $\boldsymbol{x} \models \Phi_j$ iff $\theta_{\boldsymbol{x}} \models \Phi_j^{[]}$.

If the input variable X_i is already a Boolean variable, it does not need to be re-written with this process, but for consistency of notation we map it to the partition variables $X_{[i,1]} = X_i$, and $X_{[i,2]} = \neg X_i$.

4.2 The Partition Variable Constraint Ψ

The Boolean variables $\boldsymbol{X}_{[i]}$ are subject to the constraint that exactly one of these variables is true. This is captured by the following formula:

$$\Psi_i = \left(\bigvee_j X_{[i,j]}\right)\left(\bigwedge_{j<l} (\neg X_{[i,j]} \vee \neg X_{[i,l]})\right)$$

Ψ_i is a conjunction of $2(T_i + 1)$ clauses, where T_i is the number of thresholds associated with variable X_i. Taking the conjunction over all input variables, we obtain the following constraint expression Ψ:

$$\Psi = \bigwedge_i \Psi_i \tag{1}$$

We have the following:

Lemma 10. *For any assignment θ, $\theta \models \Psi$ iff there exists values $\boldsymbol{x}$ such that $\theta_{\boldsymbol{x}} = \theta$.*

The proof is straightforward and omitted. The lemma says that any assignment satisfying the constraint Ψ corresponds to some values $\boldsymbol{x}$. Note that these values are not necessarily uniquely defined: in Example 9 if θ says that $X_{1,3}$ is true, then we know that $2.5 < X_1 < 4$, without knowing the actual value of x_1.

4.3 The Hard Constraint Φ_{SAT}

Given the observed output values $\boldsymbol{z}$ and the ground truth output values $\hat{\boldsymbol{z}}$, we define here two Boolean expressions, called VC-expressions, which state that the outputs are $\boldsymbol{z}$, or that the outputs are $\hat{\boldsymbol{z}}$ respectively:

$$\Phi^{[]} = \left(\bigwedge_{i.z_i=\mathtt{T}} \Phi_i^{[]}\right) \wedge \left(\bigwedge_{i.z_i=\mathtt{F}} \neg\Phi_i^{[]}\right) \tag{2}$$

$$\hat{\Phi}^{[]} = \left(\bigwedge_{i.\hat{z}_i=\mathtt{T}} \Phi_i^{[]}\right) \wedge \left(\bigwedge_{i.\hat{z}_i=\mathtt{F}} \neg\Phi_i^{[]}\right) \tag{3}$$

The superscript [] is intended to remind us that these expressions are over the partition variables $\boldsymbol{X}_{[]}$; we will drop the superscript from here on, unless it is needed to clarify the context. The expression Φ checks if the output is the "observed output" $\boldsymbol{z}$. The expression $\hat{\Phi}$ checks if the output is the "ground truth" $\hat{\boldsymbol{z}}$. If at least one error is detected in the output ($\boldsymbol{z} \neq \hat{\boldsymbol{z}}$) then Φ and $\hat{\Phi}$ cannot be satisfied simultaneously.

Example 11 (Example 9 continued). *Assume the observed output is $\boldsymbol{z} = \{z_1, z_2, z_3\} = \{\mathtt{T}, \mathtt{T}, \mathtt{F}\}$ and the ground truth is $\hat{\boldsymbol{z}} = \{\underline{\mathtt{F}}, \mathtt{T}, \underline{\mathtt{T}}\}$. Then (we drop superscripts []):*

$$\Phi = \Phi_1 \wedge \Phi_2 \wedge \neg\Phi_3$$
$$\hat{\Phi} = \neg\Phi_1 \wedge \Phi_2 \wedge \Phi_3$$

We give next a technical definition, which measures the distance between two assignments:

Definition 12 (Constrained Distance). *The distance between two assignments $\theta \models \Psi$ and $\hat{\theta} \models \Psi$ of $\boldsymbol{X}_{[]}$ under constraint Ψ is*

$$d(\theta, \hat{\theta}) = \frac{1}{2} h(\theta, \hat{\theta})$$

where $h(\theta, \hat{\theta})$ is the hamming distance between the two Boolean vectors $\theta(\boldsymbol{X}_{[]})$ and $\hat{\theta}(\boldsymbol{X}_{[]})$

If we have two input values $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ that differ in exactly one variable X_i, then their corresponding assignments $\theta_{\boldsymbol{x}}$ and $\theta_{\hat{\boldsymbol{x}}}$ will differ in exactly two variables, hence, by our definition, $d(\theta_{\boldsymbol{x}}, \theta_{\hat{\boldsymbol{x}}}) = 1$.

The crux of our translation is captured by the next definition and theorem. They map the notion of VC-causes of $\boldsymbol{z}|\hat{\boldsymbol{z}}$ into the a problem of "causes for Ψ, Φ, $\hat{\Phi}$".

Definition 13. *Consider the Boolean expressions Ψ, Φ, $\hat{\Phi}$ given by (Eq. 1), (Eq. 2), and (Eq. 3), and let θ be an assignment of the partition variables $\boldsymbol{X}_{[]}$. Assume the following:*

- *$\theta \models \Psi$ (it satisfies the constraints)*
- *$\theta \models \Phi$ (it generates the "observed" output)*

A variable $X_{[i,j]}$ is a cause *for Ψ, Φ, $\hat{\Phi}$ if there exist two assignments θ^Γ and $\hat{\theta}$ such that:*

- *$\theta^\Gamma \models \Psi$ and $\hat{\theta} \models \Psi$ (both satisfy the constraint)*
- *$\hat{\theta}(X_{ij}) \neq \theta^\Gamma(X_{ij}) = \theta(X_{ij})$ and $d(\theta^\Gamma, \hat{\theta}) = 1$ (they differ in only one input variable)*
- *$\theta^\Gamma \not\models \hat{\Phi}$ and $\hat{\theta} \models \hat{\Phi}$ (θ^Γ does not yet generate the ground truth, but $\hat{\theta}$ does generate the ground truth)*

We call the set $\Gamma = \{X_{[u,v]} \mid \theta^{\Gamma}(X_{[u,v]}) \neq \theta(X_{[u,v]})\}$ *a contingency of* $X_{[i,j]}$*, and* θ^{Γ} *is the contingent assignment.*

THEOREM 14 (VC CAUSE EQUIVALENCE). $X_{[i,j]}$*, with* $\theta(X_{[i,j]}) = \mathtt{T}$*, is a cause of* Ψ*,* Φ*,* $\hat{\Phi}$ *iff* X_i *is a VC-cause of* $\boldsymbol{z}|\hat{\boldsymbol{z}}$*.*

The proof is in the appendix. Thus, we reduce the problem of computing VC-causes of $\boldsymbol{z}|\hat{\boldsymbol{z}}$ to the problem of computing causes for Ψ, Φ, $\hat{\Phi}$. We reduce the latter to the satisfiability problem.

Given a variable $X_{[i,j]}$, we will construct a Boolean formula that is satisfiable if and only if $X_{[i,j]}$ is a cause of Ψ, Φ, $\hat{\Phi}$. We denote with $\hat{\Phi}\left[\theta(X_{[i,j]})\right]$ the Boolean expression that results from $\hat{\Phi}$ after assigning $X_{[i,j]}$ to its value under θ, whereas $\hat{\Phi}\left[\neg\theta(X_{[i,j]})\right]$ denotes the expression that we get by assigning to it its negation. Similarly, $\hat{\Phi}\left[\theta(\boldsymbol{X}_{[i]})\right]$ denotes the expression that results from assigning to $\hat{\Phi}$ all variables in $\boldsymbol{X}_{[i]}$ to their values under θ.

If $X_{[i,j]}$ is a cause of Ψ, Φ, $\hat{\Phi}$, then there must exist an assignment that makes the following formula `true`:

$$\Phi_{\text{SAT}} = \neg\hat{\Phi}\left[\theta(\boldsymbol{X}_{[i]})\right] \wedge \hat{\Phi}\left[\neg\theta(X_{[i,j]})\right] \wedge \Psi\left[\neg\theta(X_{[i,j]})\right] \quad (4)$$

In the above formula, $\hat{\Phi}$ denotes the Boolean formula from Eq. 3, while Ψ is the constraint expression, as defined in Sect. 4.2. The first term ($\neg\hat{\Phi}\left[\theta(X_{[i]})\right]$) ensures the requirement that $\theta^{\Gamma} \not\models \hat{\Phi}$: when $\boldsymbol{X}_{[i]}$ is set to its original values the assignment does not satisfy $\hat{\Phi}$. The second term ($\hat{\Phi}\left[\neg\theta(X_{[i,j]})\right]$) ensures that $\hat{\theta} \models \hat{\Phi}$: when X_i switches values (since $X_{[i,j]}$ is forced to its negation) the assignment should satisfy $\hat{\Phi}$. Finally, the third term enforces the constraint over the partition variables, under the assumption that the value of X_i should change.

We can state now our main result for the reduction from causality to SAT:

THEOREM 15. X_i *is a VC-cause of* $\boldsymbol{z}|\hat{\boldsymbol{z}}$ *iff* Φ_{SAT} *given by (Eq. 4) is satisfiable.*

Note that we cannot yet use a standard SAT solver to solve the satisfiability of Eq. 4, because these require the expression to be translated into CNF. Translating Eq. 4 into CNF is a non-trivial task that can result in an exponential increase in size, and we address it in Sect. 5. First we show how to extend this construction to compute the responsibility.

4.4 Computing Responsibility

We can now define responsibility:

DEFINITION 16. *Fix the Boolean expressions* Ψ*,* Φ*,* $\hat{\Phi}$ *over variables* $\boldsymbol{X}_{[]}$*. The responsibility of* $X_{[i,j]}$ *is:*

$$\rho = \frac{1}{1 + \min_{\theta^{\Gamma}} d(\theta, \theta^{\Gamma})}$$

where θ^{Γ} *is a contingent assignment of* $X_{[i,j]}$*.*

Like causality, we can also determine the responsibility of the input variables $\boldsymbol{X}$ over the Boolean partition variables $\boldsymbol{X}_{[]}$.

THEOREM 17 (RESPONSIBILITY EQUIVALENCE). *A variable* $X_{[i,j]}$*, with* $\theta(X_{[i,j]}) = \mathtt{T}$ *is a cause for* Ψ*,* Φ*,* $\hat{\Phi}$ *and has responsibility* ρ *if and only if* X_i *is a VC cause of* $\boldsymbol{z}|\hat{\boldsymbol{z}}$ *with responsibility* ρ*.*

Computing the responsibility of a cause is a harder problem, as it requires finding the minimum size contingency set Γ. Following Def. 13, that corresponds to determining the assignment $\hat{\theta}$ that differs the least from the original assignment θ, which satisfies Φ_{SAT}.

We will solve the responsibility problem by creating an instance of a partial weighted MaxSAT problem [15]. In partial weight MaxSAT each clause is given a weight, with a maximum weight identifying *hard constraints*, which are the clauses that are required to be satisfied. A solution to the problem finds an assignment that satisfies all the hard constraints, and maximizes the weight of the satisfied soft constraints.

In our case, Φ_{SAT} (Eq. 4) defines hard constraints: all of its clauses should be satisfied for a variable to be a cause. In addition we ask the solver to find an assignment that is as close as possible to θ. We do this by adding the following formula:

$$\Phi_{\theta} = \bigwedge_{\theta(X_{[i,j]})=\mathtt{T}} X_{[i,j]} \bigwedge_{\theta(X_{[i,j]})=\mathtt{F}} \neg X_{[i,j]} \quad (5)$$

Φ_{θ} represents in a Boolean expression the given assignment θ. Note that Φ_{θ} is already written in CNF, and each clause consists of exactly one variable $X_{[i,j]}$, possibly negated. We assign to each clause a weight of 1, thus asking the solver to violate as few clauses as possible. The only assignment that makes Φ_{θ} true is θ. For any other assignment $\hat{\theta}$, the number of clauses in Φ_{θ} that are false is precisely $d(\theta, \hat{\theta})$. Thus, a partial weighted MaxSAT solution will find the assignment that satisfies the hard constraints Φ_{SAT}, and differs the least from the assignment θ. We state this formally:

THEOREM 18. *Consider the MaxSAT problem consisting of the hard constraint* Φ_{SAT} *and the soft clauses in* Φ_{θ}*, and let* t *be the minimum number of clauses in* Φ_{θ} *that are false under any assignment. Then, the responsibility of* X_i *for* $\boldsymbol{z}|\hat{\boldsymbol{z}}$ *is* $\rho = 1/(1 + \frac{t}{2})$*.*

Note that t is divided by 2 in the responsibility computation due to the constraint over the partition variables: a distance of 2 in an assignment in the partition space corresponds to a single variable change in the input space.

5. CONVERSION INTO CNF

In the previous section, we described how the problem of determining view-conditioned causes and their responsibilities can be translated into a general satisfiability problem. While satisfiability is well known to be hard in general, there are highly optimized tools available that can solve satisfiability for CNF expressions [18]. Hence, we need to convert our general expression Φ_{SAT} in Eq. 4 into CNF before testing causality and responsibility for individual variables. The time complexity of CNF conversion is linear [30], but it can introduce exponential blow-ups in the size of the expression. The problem comes from the necessity to negate, at several steps, CNF formulas, but then reformulate the result as CNF. The standard way to do push the negation into the literals by applying De Morgan's laws and distributivity. There are more sophisticated conversion algorithms that can be employed (e.g. [30]), but the optimization that we propose here is orthogonal to those, and can be used in conjunction with other conversion methods. In our problem, there are two critical points in the conversion, that is expressing $\hat{\Phi}$ as CNF, and that of expressing its negation $\neg\hat{\Phi}$.

In this section we present an optimization that exploits the constraints of the Boolean partition variables in order to reduce the size of the converted CNF expression. Algorithm 1 provides a conversion that completely solves the problem for $\hat{\Phi}$ ($\hat{\Phi}$ is constructed in a way that ensures it is already in CNF) and ameliorates the size increase for $\neg\hat{\Phi}$. The algorithm exploits the fact that a threshold predicate $Y_i^{(j)}$ can either be written in the standard form given in Sect. 4, or alternatively in what we call its *dual form*. For this section, we assume that the initial classifiers are given in CNF, but the proposed optimization can be beneficial even when that is not the case. Assume we have the following $m = 2$ classifiers over $n = 3$ inputs:

$$\Phi_1 = (X_1 < 6) \wedge (X_2 \leq 8)$$
$$\Phi_2 = (X_1 < 5) \wedge (X_3 \geq 7)$$

Then the threshold predicate $Y_1^{(1)} = (X_1 < 6)$ over the partitioned variables can be expressed with either of the following two forms:

$$Y_1^{(1)} \equiv X_{[1,1]} \vee X_{[1,2]} \vee X_{[1,3]}$$
$$Y_{1D}^{(1)} \equiv \bar{X}_{[1,4]} \wedge \bar{X}_{[1,5]}$$

Here we use the optional subscript D to indicate the dual form and write $\bar{X}_i$ as short form for $\neg X_i$. When constructing Φ_{SAT}, Algorithm 1 chooses at each step an equivalent expression that minimizes the total number of clauses in the resulting CNF: Line 2 to line 4 encode the conversion for $\hat{\Phi}$ and add clauses for Φ_j or $\neg\Phi_{jD}$, depending on whether $\hat{z}_j$ is true or false, respectively. Here, again the subscript D indicates that $\neg\Phi_{jD}$ is calculated starting from the duals $Y_{iD}^{(j)}$ instead of the standard form, and becomes only one clause. For example,

$$\neg\Phi_{1D} = \big(X_{[1,4]} \vee X_{[1,5]} \vee X_{[2,3]}\big)$$

whereas starting from the standard form would result in

$$\begin{aligned}\neg\Phi_1 = &\big(\bar{X}_{[1,1]} \vee \bar{X}_{[2,1]}\big) \wedge \big(\bar{X}_{[1,1]} \vee \bar{X}_{[2,2]}\big) \wedge \\ &\big(\bar{X}_{[1,2]} \vee \bar{X}_{[2,1]}\big) \wedge \big(\bar{X}_{[1,2]} \vee \bar{X}_{[2,2]}\big) \wedge \\ &\big(\bar{X}_{[1,3]} \vee \bar{X}_{[2,1]}\big) \wedge \big(\bar{X}_{[1,3]} \vee \bar{X}_{[2,2]}\big)\end{aligned}$$

Denoting with f the number of ground truth values F, $\hat{\Phi}$ is converted into $\mathcal{O}\big((m-f)n\big)$ clauses as follows:

$$\hat{\Phi} = \underbrace{\neg\Phi_{1D} \wedge \ldots \wedge \neg\Phi_{fD}}_{f \text{ clauses}} \wedge \underbrace{\Phi_{f+1} \wedge \ldots \wedge \Phi_m}_{\mathcal{O}((m-f)n) \text{ clauses}}$$

whereas the naive conversion would lead into $\mathcal{O}(fk^n)$ clauses.

This difference becomes even more pronounced for converting $\neg\hat{\Phi}$, where an exponential blow-up is unavoidable. Here, Line 5 to line 11 first construct the following formula:

$$\neg\hat{\Phi} = \underbrace{\Phi_1 \vee \ldots \vee \Phi_f}_{\substack{\boldsymbol{\beta}\text{: } f \text{ conjuncts} \\ \text{with } n \text{ clauses}}} \vee \underbrace{\neg\Phi_{f+1D} \vee \ldots \vee \neg\Phi_{mD}}_{\boldsymbol{\alpha}\text{: 1 clause}}$$

Converting this formula to CNF results in $\mathcal{O}(n^f)$ clauses, and this exponential size increase cannot be avoided. In contrast, the naive conversion would even lead to a double exponential number of clauses.

Finally, line 12 to line 14 adds the constraint expression Ψ by adding $\mathcal{O}(mk^2)$ clauses, where k is the maximum number of partitions for any input variable.

```
**Algorithm**: CNF conversion of a VC causality problem
**Input**: Partitioned variables $\boldsymbol{X}_{[]}$, ground truth $\hat{\boldsymbol{z}}$
           Boolean expressions $\Phi_j$ and $\neg\Phi_{jD}$ for all $j$
**Output**: General $\Phi_{\text{SAT}}$ in CNF
 1  Let $\Phi_{\text{SAT}}$ be a conjunction of disjunctions (CNF)
 2  //Add clauses for $\hat{\Phi}$
 3  $\forall j \in [m]$ with $\hat{z}_j =$ T: add $\Phi_j$ to $\Phi_{\text{SAT}}$
 4  $\forall j \in [m]$ with $\hat{z}_j =$ F: add $\neg\Phi_{jD}$ to $\Phi_{\text{SAT}}$
 5  //Add clauses for $\neg\hat{\Phi}$
 6  Let $\boldsymbol{\alpha}$ be a disjunction of literals
 7  $\forall j \in [m]$ with $\hat{z}_j =$ T: add $\neg\Phi_{jD}$ to $\boldsymbol{\alpha}$
 8  Let $\boldsymbol{\beta}$ be a disjunction of conjuncts
 9  $\forall j \in [m]$ with $\hat{z}_j =$ F: add $\Phi_j$ to $\boldsymbol{\beta}$
10  Convert $\gamma \leftarrow \boldsymbol{\alpha} \vee \boldsymbol{\beta}$ from DNF into CNF with De Morgan
11  Add $\boldsymbol{\gamma}$ to $\Phi_{\text{SAT}}$
12  //Add clauses for $\Psi$
13  $\forall i \in [n]$: add $\big(\bigvee_{k \in [k_i]} X_{[i,k]}\big)$ to $\Phi_{\text{SAT}}$
14  $\forall i \in [n], j < l \in [k_i]$: add $\big(\bar{X}_{[i,j]} \vee \bar{X}_{[i,l]}\big)$ to $\Phi_{\text{SAT}}$
```

Algorithm 1 converts the problem of finding VC causes into solving a general CNF formula. The actual evaluation later replaces some clauses with actual values for variables.

6. EXPERIMENTAL EVALUATION

We validate our approach by using a real data set collected from a practical application that operates along the lines of our classification example. The application, called CARE (Context-Aware Recommendation Engine), collects various sensor data from a user's smartphone, uses it to infer the current context (e.g., whether the user is walking or driving), and provides various context-aware services. For example, CARE allows a user to search for restaurants. If CARE thinks that the user is walking alone, it will show search results or ads of restaurants within the user's walking distance from the current location. On the other hand, if it believes that the user is driving with their family, it will provide a list of family restaurants within driving distance.[1]

CARE relies on (i) a set of extractors equivalent of the pre-processing layer of Fig. 3, which extract useful features from sensor data (e.g., user's speed from GPS data) and (ii) a set of classifiers that infer the user's current context based on the values of the features. These features are used as our input variables, and all the classifier outputs comprise our views. Assuming correctness of the classifier functions, errors in the classification can still occur due to various factors, such as innate unreliability of some of the sensory input, or even unpredictable user behavior (e.g., the user is outdoors with the phone in their pocket, resulting in invalid light sensor data). CARE receives feedback on the correctness of the classifier outputs either directly from user input, or indirectly through the user response to the context-based recommendations. In cases of errors, it is important for CARE to *determine which input is responsible for the errors*, and thus ignore the corresponding classifiers in the recommendation process.

For the experiments in this section, we collect data using CARE's data collection software. We use the software on Android-based HTC Desire smartphones. When instructed, the software collects data from various phone sensors such as accelerometer, GPS, microphone, light sensor, and cell

[1]In addition to the current inferred context, CARE also uses user preferences and historical data in making such recommendations.

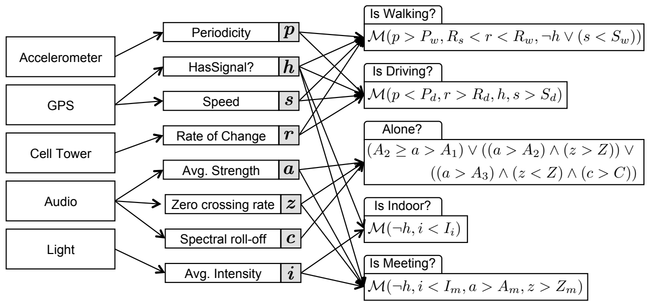

Accelerometer
GPS
Cell Tower
Audio
Light
Periodicity p
HasSignal? h
Speed s
Rate of Change r
Avg. Strength a
Zero crossing rate z
Spectral roll-off c
Avg. Intensity i
Is Walking?
$\mathcal{M}(p > P_w, R_s < r < R_w, \neg h \vee (s < S_w))$
Is Driving?
$\mathcal{M}(p < P_d, r > R_d, h, s > S_d)$
Alone?
$(A_2 \geq a > A_1) \vee ((a > A_2) \wedge (z > Z)) \vee ((a > A_3) \wedge (z < Z) \wedge (c > C))$
Is Indoor?
$\mathcal{M}(\neg h, i < I_i)$
Is Meeting?
$\mathcal{M}(\neg h, i < I_m, a > A_m, z > Z_m)$

Figure 4: Structure of our classifier system based on sensory input from a mobile device.

phone signal strength. During data collection, the software also asks the user to annotate the data with the true context (e.g., walking or not) and validity of various sensor inputs (e.g., a light sensor input is invalid if the phone is inside a bag). The data and annotations are then used to build one machine learning classifier (i.e. a decision tree) for each context of interest. The resulting classifiers with five target contexts are shown in Fig. 4.

The dataset used in the experiments in this section was collected from three individuals, who carried our instrumented smartphones over a period of three weeks and collected data during their daily activities. The dataset contains in total 801 different contexts, spanning a time range of over 149 hours in a three-week period. Each of the contexts in the dataset has its start and end times, associated data from all sensors, its true label, the label given by classifiers, and the validity of each sensor input. Thus, for each case of misclassification, we use the sensor validity as the ground truth for causality against which we validate our approach.

Implementation. Our satisfiability reduction algorithm was implemented in Java.[2] For every instance of the input data, our algorithm constructs the appropriate data files in standard DIMACS CNF and WCNF format [21]. We use the `minisat` [32] and `MiniMaxSat` [18] solvers, to solve the SAT and partial weighted MaxSAT instances, respectively.

6.1 Effectiveness

In our experiments we want to validate the quality of information of the view-conditioned responsibility ranking over possible sources of error. We ran our algorithms over the test data collected through the CARE system, and each test case took only a couple of seconds to complete. We first evaluate the effectiveness of our approach compared to other schemes using average precision as the metric of comparison [23]. *Average precision* (AP) quantifies the accuracy of a ranking in identifying points of relevance. Given the ground truth on the validity of the sensory input, an average precision of 1 means that the faulty sensors were indeed ranked first in the ranking of most likely errors.

We compare view-conditioned causality against three other techniques that can be used to produce error rankings: (*i*) Boolean variable influence, (*ii*) view-conditioned counterfactuals, and (*iii*) causality without conditioning. *Boolean influence* produces a *static ranking* of the input parameters for each classifier. The influence of a variable X_i in formula Φ is defined as the probability that Φ remains undetermined when values are assigned to all variables except X_i [24]. For example, in the simple formula $\Phi = A \vee (B \wedge C)$, the influence of A is 0.75, whereas the influence of B is 0.25. Note that influence quantifies the importance of a variable within a function over all possible assignments. On the other hand, causality-based schemes quantify this importance based on the actual assignment. Intuitively, we expect schemes that take into account the current state of the world to be more accurate in error identification.

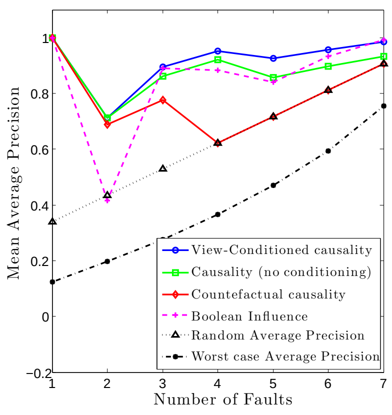
Figure 5a shows the results of this first set of experiments. The horizontal axis shows the number of faults in the input parameters, starting from a single parameter failing, and going up to 7 (recall from Fig. 4 that the classifiers operate over 8 parameters in total). Causality-based schemes are depicted with solid lines. Also depicted are random average precision and the worst case average precision for each fault range.[3] Note that view-conditioned causes are always a better predictor for errors than view-conditioned counterfactuals or causes without conditioning. And this difference gets more pronounced with the number of faults. Also note that that the precision of counterfactual causes follows that of random ranking from a point on. It is normal because as the number of faults increases, we stop having any counterfactual causes at all.

The dip observed across all schemes at the instances with 2 faults is due to the lack of granularity in the ground truth of sensor validity. Our ground truth is restricted to the level of the sensors (e.g. GPS), rather than the individual parameters (e.g. h). So when the GPS is considered faulty, the features h and s are both deemed incorrect. However, with this few errors, only one or two classifiers are wrong, and they are likely to only include h (not s). The result is that s is not really a cause (it is not part of the classifiers that failed), but because it is always considered to fail with h in the ground truth, it is reflected as loss in precision.

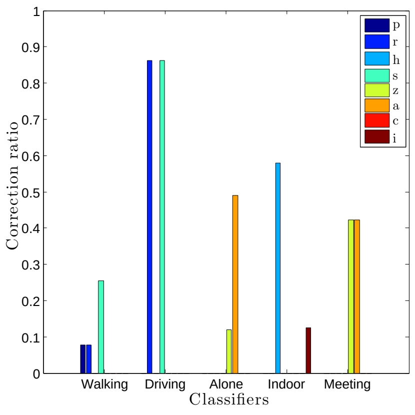
In Figure 5b we studied the next step: how well can we correct errors in the classifiers, once the likely input errors are identified. The graph displays the portion of errors in the particular classifier that can be corrected if the top ranked view-conditioned cause is omitted from the calculation of the classifier result. This involves a modification in the classifier function. For example, the driving classification is a strict majority function between 4 predicates; if feature p is omitted, then it becomes a strict majority among the other three predicates. An interesting observation in our experiments is that certain classifiers can achieve better correction ratios than others. For example, omitting features r or s corrects more than 85% of the errors of the driving classification, whereas omitting parameter s corrects the walking classifier by only 25%. This behavior may seem random or surprising at first, but in fact it highlights a very important insight that causality brings into error correction. This can be understood from Fig. 5c, which focuses on three of the classifiers (walking, driving, and meeting) to explain the reasons for this difference in corrections: Note that the driving classification, as seen in Fig. 4, uses 4 features (p, r, s and h), two of which (r and s) are counterfactual ($\rho = 1$ in Fig. 5c), whereas the other two have zero responsibility. This is very

[2]In practice, the algorithm would be running on the phone itself. If the implementation is heavy weight, it can run on the Cloud and the phone can communicate with it when needed.

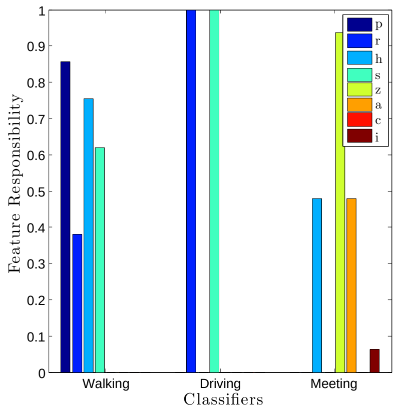
[3]Random AP is the expected AP when a list of k relevant and n total items is randomly sorted. Worst case AP is when those k are sorted last in the list of n total items.



(a) Mean average precision (AP)



(b) Ratio of Corrections



(c) Responsibilities per parameter

Figure 5: (a): Mean average precision across different approaches: View-conditioned causality outperforms the static and simpler causality schemes. (b): Portion of errors that get corrected when the parameter with the highest responsibility is omitted from the computation. (c): Comparison among the responsibility rankings for different classifiers. (b & c): The ratio of correction actually depends on whether there exist variables *reliable* for the classifier (compare walking and driving).

important, as it means that the parameters h and p are very unlikely to cause errors in the driving classification, and are thus reliable predictors for it. Thus, when omitting r or s from the classifier function, the result becomes much more reliable. On the other hand, the walking classifier does not have reliable features, as all of them have fairly high responsibility across our test cases. When all features of a classifier have high responsibility, it means that all of them are likely causes of error. Without a single reliable predictor, it is less likely that we can achieve corrections by simply removing a parameter and not changing the classifier function otherwise. The meeting classifier stands on a middle ground: it has one reliable predicate (based on parameter i), but still contains 3 unreliable ones resulting in less gain compared to the driving classification. Hence, using responsibility to determine the reliability of features is very important in this application, as it can *quantify the quality of a classifier*, and implies possible ways to improve it. For example, in the driving classification, the vast difference between the reliable and unreliable features is an indication that parameters r and s should be disregarded in this classifier altogether, whereas the absence of reliable predicates for walking likely means that the learned models need to be improved.

6.2 Scalability

In our second set of experiments we study how our approach performs for bigger problem sizes. We generate synthetic data with a similar structure to the one in CARE. We randomly generate classifier functions as a conjunction over ranges of the different parameters, and randomly assign classification failures with a probability of about 10%, following the failure rate that we observed in the real data. In the context of an application like the CARE system we do not expect to have more than 15-20 parameters/features and 20-30 classifiers. Yet in this evaluation, we generate instances from 5 to 70 parameters, and from 5 to 60 classifiers.

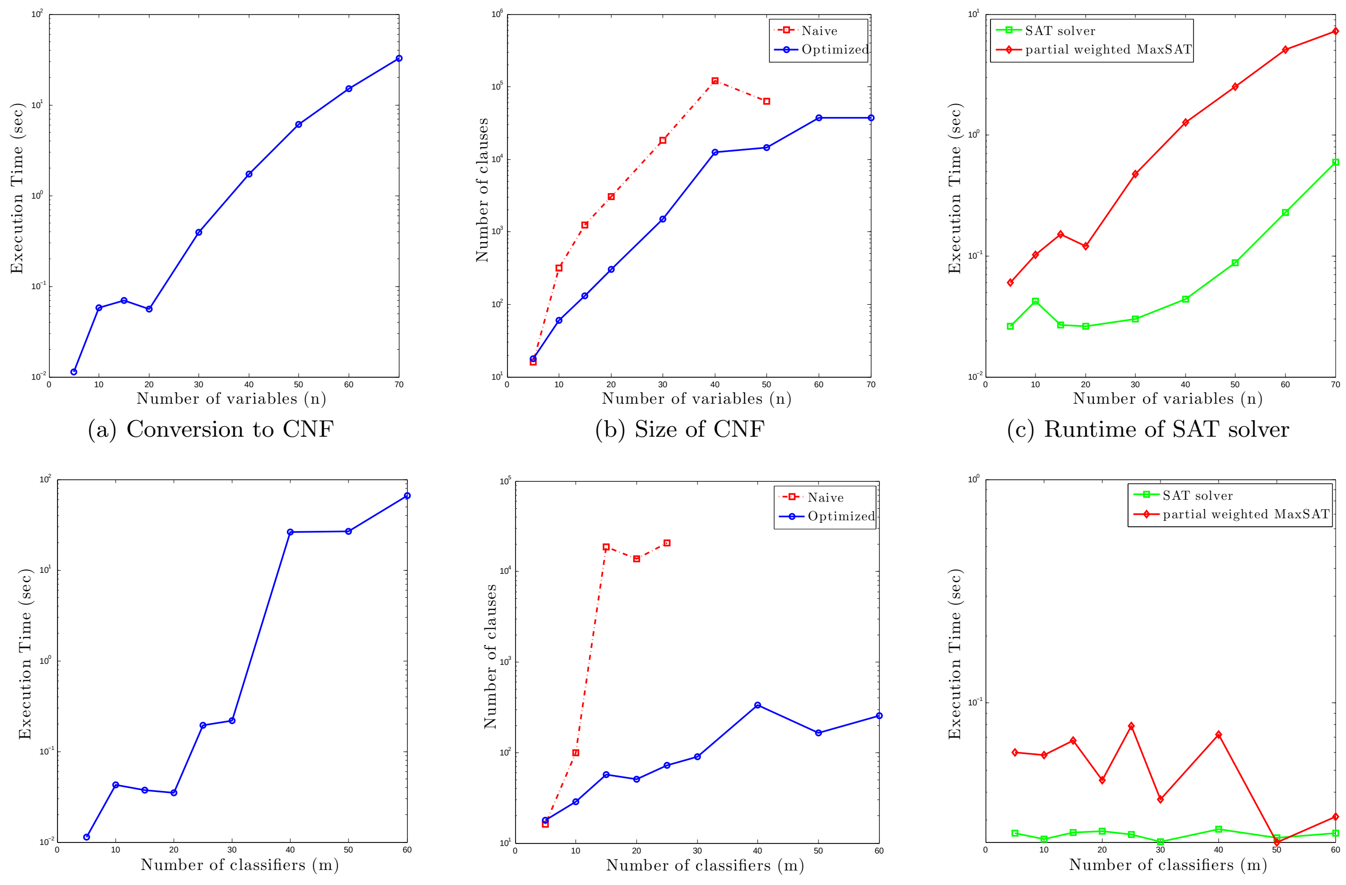
Figure 6a and Fig. 6d show the execution time of our CNF conversion algorithm 1, which takes as input the classification formulas, variable assignments and observed errors, and produces instances of SAT and partial weighted MaxSAT. Note that the vertical axis of these graphs is in log scale. As discussed in Sect. 5, conversion of the computed formulas into CNF is required in order to be solved by standard satisfiability tools. As the problem size grows with more variables or classifiers, thus generating larger formulas, our algorithm is more likely to suffer the effects of exponential growth during the necessary CNF conversion. Note however that this conversion is not an inherent part of our reduction from view-conditioned causality to SAT, which is polynomial in size, but rather a complication imposed by the CNF requirement of currently available SAT solvers. In spite of the CNF conversion step, the runtime of our algorithm remains feasible with average runtime of around one minute for the larger problem sizes, and around one second for sizes of parameters and classifiers what an application along the lines of CARE would actually need or have available.

Figure 6b and 6e show that our optimized conversion produces CNF formulas of at least an order of magnitude smaller than a naive conversion. Smaller expression sizes are desirable, as on expectation they are likely to improve both the runtime of our own conversion algorithm, and the execution times of the SAT solvers.

Through these experiments, we also verify the rationale of our initial premise, which was to reduce view-conditioned causality to a satisfiability problem in order to be able to use existing efficient tools to solve it. Even though satisfiability is hard in general, current tools can very efficiently handle a large space of problems. Figure 6c and 6f demonstrate the execution times of the SAT solver (`minisat`) and partial weighted MaxSAT solver (`MiniMaxSat`), which remain highly competitive. In our problem structure, it seems that the solvers are barely affected by the number of classifications with their runtime kept well below a second. When increasing the number of parameters, the `minisat` runtime remains fairly stable. The runtime of `MiniMaxSat` does increase, but is still under 7sec for our larger problem size.

7. CONCLUSIONS

In this paper we considered data transformations that map a source database to a target database, and studied the problem of tracing errors detected in the target data



(a) Conversion to CNF (b) Size of CNF (c) Runtime of SAT solver

(d) Conversion to CNF (e) Size of CNF (f) Runtime of SAT solver

Figure 6: Scalability when varying the number n of input variables (a-c), and the number m of classifiers (d-f), averaged over $N = 10$ random input problems. As expected, the runtime of converting a view-conditioned causality problem to a CNF satisfiability problem is exponential in n (a) and m (d). However, the size of the CNF produced by our optimized conversion is considerably smaller than a naive conversion (b,e). The runtime of the SAT solvers is exponential in n (c), but for the structure of the formulas in our setting it does not seem to be affected by m (f).

back to the source data. Our source data is described by a set of numerical or Boolean variables, the target data is given by a set of Boolean variables, and the transformation is given by a set of Boolean expressions. We proposed *view-conditioned* causality and responsibility as a solution to trace errors from transformations to source data. Next, we described an algorithm that reduces the view-conditioned responsibility problem to general satisfiability, which can be solved with existing tools. We also illustrated the effectiveness of using our techniques for tracking, and even correcting errors in a real application.

We believe that the data-level transformations studied in this paper can be improved in future work by considering the query that generated the transformations. By taking into account the query expression it may be possible to extend our algorithm for view-conditioned responsibility to handle larger scale transformations, as found in data integration and ETL tools.

8. REFERENCES

[1] http://www.satcompetition.org/.

[2] T. Alsinet, F. Manyà, and J. Planes. Improved exact solvers for weighted Max-SAT. In *SAT*, pp. 371–377, 2005.

[3] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, pp. 1–12, 2007.

[4] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, vol. 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[5] P. Buneman, J. Cheney, W. C. Tan, and S. Vansummeren. Curated databases. In *PODS*, pp. 1–12, 2008.

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

[7] P. Buneman, S. Khanna, and W. C. Tan. On propagation of deletions and annotations through views. In *PODS*, pp. 150–158, 2002.

[8] A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pp. 523–534, 2009.

[9] J. Cheney. Causality and the semantics of provenance. In *DCM*, pp. 63–74, 2010.

[10] 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.

[11] H. Chockler and J. Y. Halpern. Responsibility and blame: A structural-model approach. *J. Artif. Intell. Res. (JAIR)*, 22:93–115, 2004.

[12] G. Cong, W. Fan, F. Geerts, and J. Luo. On the

complexity of view update and its applications to annotation propagation. *TKDE*, 2011.

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

[14] T. Eiter and T. Lukasiewicz. Complexity results for structure-based causality. *Artif. Intell.*, 142(1):53–89, 2002. (Conference version in *IJCAI*, 2002).

[15] Z. Fu and S. Malik. On solving the partial Max-SAT problem. In *SAT*, pp. 252–265, 2006.

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

[17] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *Brit. J. Phil. Sci.*, 56:843–887, 2005. (Conference version in *UAI*, 2001).

[18] F. Heras, J. Larrosa, and A. Oliveras. MINIMAXSAT: an efficient weighted Max-SAT solver. *J. Artif. Int. Res.*, 31(1):1–32, 2008.

[19] M. Herschel and M. A. Hernández. Explaining missing answers to SPJUA queries. *PVLDB*, 3(1):185–196, 2010.

[20] M. Herschel, M. A. Hernández, and W. C. Tan. Artemis: A system for analyzing missing answers. *PVLDB*, 2(2):1550–1553, 2009.

[21] H. H. Hoos and T. Stützle. SATLIB: An online resource for research on SAT. pp. 283–292. IOS Press, 2000.

[22] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.

[23] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, pp. 41–48, 2000.

[24] J. Kahn, G. Kalai, and N. Linial. The influence of variables on Boolean functions (extended abstract). In *FOCS*, pp. 68–80, 1988.

[25] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, pp. 61–75, 2005.

[26] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pp. 233–246, 2002.

[27] D. Lewis. Causation. *The Journal of Philosophy*, 70(17):556–567, 1973.

[28] 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.

[29] P. Menzies. Counterfactual theories of causation. Stanford Encylopedia of Philosophy, 2008.

[30] D. Sheridan. The optimality of a fast CNF conversion and its use with SAT. In *SAT*, 2004.

[31] A. Simitsis, P. Vassiliadis, and T. K. Sellis. Optimizing ETL processes in data warehouses. In *ICDE*, pp. 564–575, 2005.

[32] N. Sorensson and N. Een. Minisat v1.13 – a SAT solver with conflict-clause minimization. In *SAT 2005 competition*, 2005.

[33] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD*, pp. 15–26, 2010.

APPENDIX

A. PROOFS

PROOF (THEOREM 14). Let $X_{[i,j]}$ be a VC cause of $\boldsymbol{z}|\hat{\boldsymbol{z}}$ with responsibility ρ. Then $\exists\theta^{\Gamma}$ as per the conditions of Def. 13. Assign $\Gamma' = \{X_{[u,v]}\,|\,\theta^{\Gamma}(X_{[u,v]}) \neq \theta(X_{[u,v]})\}$. Γ' contains exactly those variables $X_{[u,v]}$ that changed their assignment between θ and θ^{Γ}. Since $\theta^{\Gamma} \models \Psi$, for each $X_{[u,v_1]} \in \Gamma'$ there exists a second $X_{[u,v_2]} \in \Gamma'$ corresponding to the same variable X_u. Let Γ be the set $\{X_u\,|\,\boldsymbol{X}_{[u]} \cap \Gamma' \neq \emptyset\}$. Then $|\Gamma| = \frac{|\Gamma'|}{2} = d(\theta, \theta^{\Gamma})$. Since $\theta^{\Gamma} \not\models \hat{\Phi}$, we know that Γ is not counterfactual for $\boldsymbol{z}|\hat{\boldsymbol{z}}$. But $\hat{\theta} \models \hat{\Phi}$, which means that $X_i \cup \Gamma$ is counterfactual, and therefore X_i is a cause with contingency Γ. We will now prove the reverse. Let X_i be a cause of $\boldsymbol{z}|\hat{\boldsymbol{z}}$. Then there exists set Γ such that $X_i \cup \Gamma$ is counterfactual for $\boldsymbol{z}|\hat{\boldsymbol{z}}$. That means that there exist values $\hat{x}_u$ for each $X_u \in \Gamma$, different from the original values, and $\hat{x}_i \neq x_i$, that switch the output to $\hat{\boldsymbol{z}}$. Also, $X_i \cup \Gamma$ is minimal, which means that none of the value changes are redundant. Therefore, each value change $x_u \to \hat{x}_u$ results in value changes for two partition variables $X_{[u,v_1]}$ and $X_{[u,v_2]}$. Let θ^{Γ} be the assignment of the partition variables after we apply all the value changes $x_u \to \hat{x}_u$. Since Γ is not counterfactual, we know that $\theta^{\Gamma} \not\models \hat{\Phi}$. Let $\hat{\theta}$ be the Boolean assignment after we apply the value change $x_i \to \hat{x}_i$ to θ^{Γ}, which results in two changes in the partition variables $\boldsymbol{X}_{[i]}$: $X_{[i,j]}$ switching from `T`to `F`, and some other variable in $X_{[i]}$ switching from `F`to `T`. Since $X_i \cup \Gamma$ is counterfactual for $\boldsymbol{z}|\hat{\boldsymbol{z}}$, we know that $\hat{\theta} \models \hat{\Phi}$. So, $X_{[i,j]}$ is a VC cause for $\boldsymbol{z}|\hat{\boldsymbol{z}}$. $\square$

PROOF (THEOREM 15). Let X_i be a VC cause. Then, from Theorem 14 variable $X_{[i,j]}$ is a VC cause and there exists assignment $\theta^{\Gamma} \models \Psi$, such that $\theta^{\Gamma} \not\models \hat{\Phi}$, and $\hat{\theta} = \theta^{\Gamma} \cup \{X_[i,j] = \texttt{F}\} \models \hat{\Phi}$. But that means that θ^{Γ} satisfies Φ_{SAT}. Now let $\theta^{\Gamma} \models \Phi_{\text{SAT}}$. Then $\theta^{\Gamma} \models \neg\hat{\Phi}\left[\theta(\boldsymbol{X}_{[i]})\right] \Rightarrow \theta \not\models \hat{\Phi}$, $\theta \models \Psi$, and $\theta \models \hat{\Phi}\left[\neg\theta(X_{[i,j]})\right] \Rightarrow \theta^{\Gamma} \cup \{X[i,j] = \texttt{F}\} \models \hat{\Phi}$. But then θ^{Γ} is a contingent assignment for $X_{[i,j]}$, and therefore X_i is a cause. $\square$

PROOF (THEOREM 17). The result is straightforward from the proof of Theorem 14: we have $|\Gamma| = \frac{|\Gamma'|}{2} = d(\theta, \theta^{\Gamma})$, which means the responsibility ρ is the same. $\square$

PROOF (THEOREM 18). Let θ^{Γ} be the assignment that is the solution to the partial weighted MaxSAT. Then θ^{Γ} violates the minimum number of clauses (since all soft constraints have the same weight). If θ is the original assignment, t is the number of clauses that θ^{Γ} violates, then $d(\theta, \theta^{\Gamma}) = \frac{t}{2}$. $\square$

B. NOMENCLATURE

$\boldsymbol{X}$	Set of input parameters: $\boldsymbol{X} = (X_1, \ldots, X_n)$
$\boldsymbol{x}$	Actual value assignment for input parameters: $\boldsymbol{x} = (x_1, \ldots, x_n)$ with $x_i \in D_i$ and D_i its domain
n	Number of input variables
$\boldsymbol{X}_{[n]}$	Boolean partitions of input variable X_n: $\boldsymbol{X}_{[n]} = (X_{[n,1]}, \ldots, X_{[n,k_n]})$
$\boldsymbol{x}_{[n]}$	Actual value assignment for partitions of input variable X_n: $\boldsymbol{x}_{[n]} = (x_{[n,1]}, \ldots, x_{[n,k_n]})$ with $x_{[n,i]} \in \{\texttt{T}, \texttt{F}\}$ and exactly one $x_{[n,j]}$ true, the rest are false
k_n	Number of partitions for input variable X_n
$\boldsymbol{X}_{[]}$	Set of all partitioned input variables: $\boldsymbol{X}_{[]} = (X_{[1,1]}, \ldots, X_{[1,k_1]}, X_{[2,1]}, \ldots X_{n,1}, \ldots X_{[n,k_n]})$
$\boldsymbol{Z}$	Set of Boolean output variables: $\boldsymbol{Z} = (Z_1, \ldots, Z_m)$
$\boldsymbol{z}$	Observed values of classifiers: $\boldsymbol{z} = \{z_1, \ldots, z_m\}$
$\hat{\boldsymbol{z}}$	Actual ground truth for outputs: $\hat{\boldsymbol{z}} = \{\hat{z}_1, \ldots, \hat{z}_m\}$
m	Number of output variables (classifiers)
Φ_m	Boolean expression for classifier C_m
$\boldsymbol{Y}^{(m)}$	Set of threshold predicates for classifier C_m, e.g. $Y_n^{(m)} \equiv (X_n > t_n^{(m)})$
t_i	One of several thresholds for variable X_i
T_i	Number of threshold values for variable X_i
θ	Boolean assignment of $\boldsymbol{X}_{[]}$