CurrentClean: Spatio-temporal Cleaning of Stale Data

Mostafa Milani
McMaster University
mmialni@mcmaster.ca

Zheng Zheng
McMaster University
zhengz13@mcmaster.ca

Fei Chiang
McMaster University
fchiang@mcmaster.ca

***Abstract*—Data currency is imperative towards achieving up-to-date and accurate data analysis. Data is considered current if changes in real world entities are reflected in the database. When this does not occur, stale data arises. Identifying and repairing stale data goes beyond simply having timestamps. Individual entities each have their own update patterns in both *space and time*. These update patterns can be learned and predicted given available query logs. In this paper, we present *CurrentClean*, a probabilistic system for identifying and cleaning stale values. We introduce a spatio-temporal probabilistic model that captures the database update patterns to infer stale values, and propose a set of inference rules that model spatio-temporal update patterns commonly seen in real data. We recommend repairs to clean stale values by learning from past update values over cells. Our evaluation shows CurrentClean's effectiveness to identify stale values over real data, and achieves improved error detection and repair accuracy over state-of-the-art techniques.**

I. Introduction

Data quality is evaluated according to a set of quality dimensions that define desirable properties the data and schema must satisfy. These dimensions include accuracy, completeness, consistency, reliability and currency [1]. *Data currency* is defined as how promptly the data is updated to reflect actual changes in the real world [1]. The proliferation of cyber-physical systems with sensors that regularly collect data for monitoring, rely on the presence of current data for timely and accurate decision making. These new IoT applications perform analytics over small data stores containing only the most recent data. Data currency continues to play a vital role in traditional business intelligence applications where up-to-date values are critical for accurate forecasting and reporting.

Given the importance of data currency for accurate analytics, there has been minimal work studying data currency. Existing data cleaning solutions have focused on improving consistency via constraint-based repairs [2], coupling with master data for improved reliability [3], capturing semantics to maximize accuracy [4], [5], holistic data cleaning systems to improve completeness [6], and human-in-the-loop systems to involve user feedback and the crowd [7]. A surprisingly large percentage of real data is *stale*. A preliminary evaluation using two real datasets containing sensor readings in an IoT smart city [8], and retail transactions [9] revealed, respectively, 20% and 16%, of the data values were stale. Out-of-date or *stale* data occurs when changes to an entity in the real world are not captured in the database due to missing or incorrect updates. For example, the latest temperature sensor reading is not recorded, or an increase in temperature did not lead to an expected change in humidity. Missing updates occur due to system malfunction, limited access control, and human error.

SRV	TMP$_1$	TMP$_2$	HMD	AIR	KW	AR
S_1	20	21	42	none	1	$A3$
S_2	28	29	40	high	5	$A2$
S_3	19	22	43	none	3	$A1$
S_4	26	32	57	none	2	$A2$

TABLE I: Monitoring metrics.

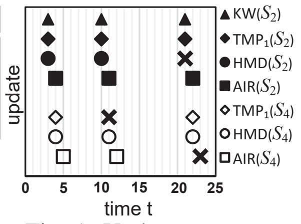


Fig. 1: Update patterns.

Missing updates for an entity are often dependent on the *spatial* correlations between updated values, and the *temporal* update frequency for that entity. We argue that data currency is a *relative* notion based on *individual* spatio-temporal update patterns. For example, the co-occurrence of updates, and the rate of updates among bank account holders vary based on individual deposit and withdrawal patterns; i.e., having the last update as of three days ago may be considered current for Alice's account but not Jane's account. Hence, quantifying currency requires identifying missing updates according to the rate of update, and the co-occurrence of past updates, according to an individual behavior pattern.

Example 1.1: Table I shows environmental sensor measurements for servers in a data center. Having current values is essential in monitoring systems to avoid operational disruption or downtime. Accurate monitoring also enables adaptive cooling in data centers to minimize energy costs. These sensor measurements include temperature readings at the bottom (TMP$_1$) and top (TMP$_2$) of server racks, humidity (HMD), air flow (AIR), kilowatt power usage (KW), and the area where a server is located (AR). Table I shows the most recent values of these sensor measurements.

Spatial Correlations. In the presence of timestamps, existing work claim that currency is trivially determined [10]. However, we argue that currency for a value is influenced by its past *updates in space and time*. First, consider spatial updates from neighboring cells within a tuple. Increased server workloads manifest in increased power usage, which trigger higher temperature and humidity levels. Consider the change in KW of S_2 (at $t = 3, 10, 21$) in Figure 1. Figure 1 shows this spatial update propagation where updates to KW co-occur at the same time with S_2 TMP$_1$ and HMD updates. The propagated update for HMD in S_2 is lost at $t = 21$ (marked with an 'X') causing S_2 HMD to be stale at the current time ($t = 25$). The update history in Figure 1 is available via database transaction logs, or flashback recovery features that allow users to view past states of database objects. Table I highlights stale values with a black background.

Temporal Correlations. Our discussion thus far considers correlated updates within a time unit. Consider temporal correlations across time, where an increase in server S_2 TMP_1 causes an increase in TMP_1 of a nearby server S_4 located in the same area $A2$. Figure 1 shows this behaviour, as an update in TMP_1 (for S_2) at $t = (3, 10, 21)$ is followed by an update to TMP_1 in S_4 during the next time unit. Similarly, the increase in S_2 TMP_1 activates the cooling unit, and consequently increases the air flow (AIR in S_2). These updates highlight chains of updates involving values within a tuple, across tuples, and across time that influence the currency of a value. Figure 1 shows missing updates (noted with 'X') that were expected based on the spatial and temporal correlations observed in the past. Note that not all missing updates translate to stale values. For example, the missing S_4 TMP_1 update at $t = 11$ is followed by an update at $t = 22$, making it current at $t = 25$. If missing updates continue until the current time, this leads to stale values, as shown in Figure 1 for S_2 HMD, and S_4 AIR.

State-of-the-Art. Repairing these stale values requires learning the spatial and temporal relationships among the cell *values*. State-of-the-art solutions have explored data cleaning over time-series data based on the rate of change of an individual cell [11], [12], and cleaning imprecise timestamps [13]. Much of this prior work identifies anomalies within an attribute (w.r.t. time), or within a single tuple or relation. Recent systems have proposed probabilistic models that learn from clean distributions in the data to minimally repair dirty values [14], [15]. HoloClean uses probabilistic inference to combine signals from multiple sources (external knowledge bases, statistical methods, data dependencies) to determine the most likely repair [6]. These techniques focus on non-temporal models of inference over a static data instance, where cleaning is irrespective of currency [6], [14], [15].

Recent work propose currency constraints that express currency relationships based on the data semantics [16]. These constraints are given apriori, and elicit a partial currency order among temporal information in the data. For example, a currency constraint states that if a person's status is "retired" and "deceased" in tuples r_1 and r_2, respectively, then r_1 precedes r_2. Currency constraints rely on domain expertise, and ignore temporal and spatial correlations.

In this paper, we address the question: *How can we detect stale values according to the spatial and temporal update patterns for a database cell, and repair these cells to current values in a unified data cleaning framework?*

Contributions. We present *CurrentClean*, a probabilistic system for detection and cleaning of stale data. CurrentClean learns spatio-temporal update patterns for a database cell via past update queries. Our work aligns with recent trends that study the value-history of individual entities [17], and proposals for change exploration systems [18], and is complementary to systems like HoloClean [6], by providing data currency detection and cleaning. We make the following contributions:

- We present: (i) a spatio-temporal update model to detect stale values based on learning from past update patterns; and (ii) a repair model and algorithm that performs inference over values to repair stale cells. (Sections II-III)
- We propose inference rules that model causal and co-occurrence update patterns to estimate currency, and to recommend spatio-temporal aware repairs. (Section IV)
- We define two types of repairs: *most-likely*, and *bounded-cost* repairs that suggest current (clean) values based on observed update values in the past. (Section V)
- We present three optimizations that prune weakly correlated updates, and improve the inference runtime. (Section VI)
- We conduct an extensive evaluation of CurrentClean's repair effectiveness, and its comparative accuracy to detect stale values in real data. We demonstrate the prevalence of our modeled update patterns, and show that CurrentClean outperforms four existing approaches by an average +34% recall to identify stale cells, and +28% F1-score in repair accuracy. (Section VII)

II. The CurrentClean Framework

We formalize our problem definition for error detection and data repair, and present an overview of CurrentClean.

A. Problem Definition

A relation R with attributes $A_1, ..., A_n$ is a finite set of n-ary tuples $\{r_1, ..., r_N\}$. A database D is a finite set of relations $R_1, ..., R_m$. We define a *cell* $r[A_i]$ as the i-th position in tuple r in D, and v_c as the value in cell c. We use $|D|$ to refer to the number of cells in D.

Given a database D and a query history $H = \langle q_1, q_2 ..., q_{|H|}\rangle$, each $q_i = (t, r[A], \mathsf{a}, \mathsf{b}) \in H$ is a query that updates the value of $r[A]$ from "a" to "b" at time t. The update history H is external information that may be obtained from sources such as database transaction logs, database flashback recovery features, or by computing the difference between successive data snapshots over a time period. The queries in H are ordered in decreasing time t. We obtain D from an initial instance D_0 by applying the updates in H. We assume a query updates a single cell, and there are no tuple insertions and deletions, and foreign key references are fixed. We assume queries in H are correct, but H may be incomplete, i.e., there may be missing updates. Let $\mathbb{H}$ be the ground truth sequence of updates, then $H \subseteq \mathbb{H}$. Let q_c and q_c^* be the last update query over cell c in H and $\mathbb{H}$, respectively. We consider c to be *current* if $q_c.t = q_c^*.t$, otherwise it is *stale*. Intuitively, c is stale if the update $q_c^* \notin H$.

Error Detection Problem. Since $\mathbb{H}$ and q_c^* are not always known, we take a probabilistic approach to estimate the currency of c based on inputs D and H. Let T_c be a random variable representing the last time when c is updated. Let p denote the probability of c being up-to-date, i.e., $P(q_c.t = T_c|D, H)$. Given D and H, the error detection problem is to

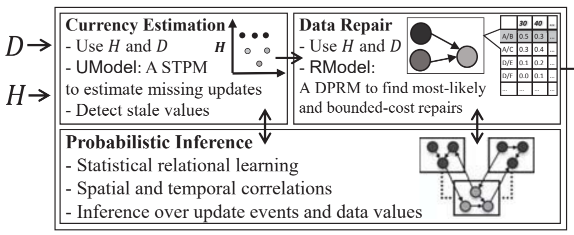


Fig. 2: CurrentClean system overview.

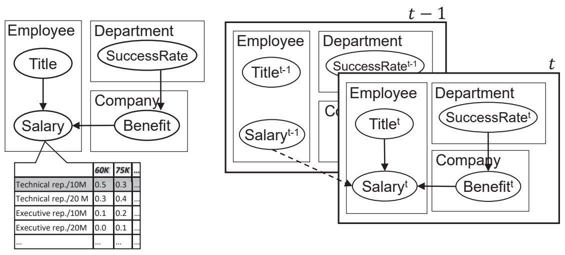


Fig. 3: (a) PRM CPD (b) 2PRM.

find a set $\mathcal{E}$ of erroneous (stale) cells in D with $p < \beta$, for a currency threshold β.

Data Repair Problem. Let v_c^* be the unknown true current value of c, and v_c' be the estimation of v_c^*. The data repair problem is to compute the value of v_c' for all cells in $\mathcal{E}$, and to generate a clean database D^C, where each v_c is replaced with its latent value v_c' as a repair. We consider a data repair to be correct if $v_c' = v_c^*$.

B. Solution Overview

Figure 2 shows the CurrentClean system architecture. Given a database D and an incomplete history H, CurrentClean returns a clean (current) database D^C. CurrentClean's execution proceeds along the following three modules.

Spatio-temporal Inference. We propose a *Spatio-Temporal Probabilistic Model* (STPM) (Section III-B) that extends a class of graphical models to capture spatial and temporal correlations between update events. The STPM model associates a random variable E_c^t to denote the occurrence of an update on a cell c at time t, and generates a probabilistic graphical model to capture the spatial and temporal relationships among all random variables for all cells in D. The STPM captures temporal and spatial locality based on the *principle of locality* (also referred to as *locality of reference*) [19]. In temporal locality, recently updated cells are likely to be updated again in the near future, and similarly, spatial locality indicates updated cells will trigger further updates to nearby cells. We instantiate a version of STPM called the UModel for currency estimation, to model updates over cells (Section III-C), and define a set of inference rules that model spatio-temporal update patterns (Section IV). In our current implementation, we use DeepDive [20] for probabilistic inference, however, CurrentClean is amenable to other probabilistic inference engines.

Currency Estimation. We generate the UModel that learns the update patterns in H and gives the joint probability distribution of the update events over cells in D during the time covered in H. Given the update history for a cell c, we compute the marginal probability of not having an update event over c from the last observed update time until the current time. This gives the probability of c being up-to-date. Cell c is stale if this probability value is less than a currency threshold β.

Data Repair. To repair D, CurrentClean generates a probabilistic repair model RModel, which is an instance of a dynamic probabilistic relational model that captures temporal and spatial patterns over the *values* in D. The RModel performs statistical learning and inference over the joint distribution of random variables V_c^t across all cells in D, where V_c^t is the random variable of cell c's value at time t. CurrentClean computes repairs via Maximum a Posteriori (MAP) inference to estimate the values of stale cells in $\mathcal{E}$. For every stale cell c_{err}, its *most-likely repair* is $u \in dom(c_{err})$ that maximizes $P(V_{c_{err}}^t = u|H, D)$. A *bounded repair* restricts the magnitude of change to be within a bound η.

III. Identifying Stale Data

We introduce a spatio-temporal probabilistic model (STPM) for estimating data currency to identify stale values. We first review a class of models that underlie the STPM foundation.

A. Dynamic Probabilistic Relational Models

A Probabilistic Relational Model (PRM) is a graphical model for inference over relational data [21]. PRMs model dependencies between attributes under a given schema, inducing probability distributions for instantiations of the schema. A PRM is a directed acyclic graph (DAG), where nodes represent relational attributes with incoming edges from its parents. Parents might be from the same relation or from another relation via a foreign key. The set of parameters in a PRM are conditional probability distributions (CPDs) that specify the probability of each attribute value given the parent values. PRMs consider objects and the links between them as first-class citizens. This allows for *structural learning* to predict the link structure, and *parameter learning* to infer the attribute values. Figure 3(a) shows a PRM with four probabilistic nodes. The CPD table shows the probabilities for Employee.Salary given parent node values, e.g., Pa(Employee.Salary) = {Employee.Title, Company.Benefit}, where Pa(X) are the parent nodes of node X.

PRMs are limited to static instances and unable to model dynamic systems that vary over time. A *Dynamic Probabilistic Relational Model* (DPRM) extends a PRM by considering instantiations of a database D at time t denoted as D^t [21]. A DPRM for D represents the probability distribution over past databases $D^0, ..., D^T$, until the current time T ($D^T = D$). DPRMs exhibit the Markovian property where each state is dependent only on the previous state, and the transition model is identical for all time slices. This is a common assumption that simplifies dynamic models and captures relationships between adjacent time slices, and some relationships involving non-adjacent time slices via transitivity among adjacent time slices. Relaxing this assumption leads to infeasible inference over a combinatorial explosion of relationships between every possible pair of time slices [22]. A DPRM is defined using two PRMs: (i) $\mathcal{M}_0$ is an initial PRM defining the distribution P over D^0, and (ii) $\mathcal{M}^{\rightarrow}$ is a *two-time-slice PRM* (2TPRM) that connects successive instances D^t to D^{t-1}. Figure 3(b) shows a DPRM defined via a 2TPRM representing the transition probability $P(D^t|D^{t-1})$, where an employee's salary at time t is dependent on her title, benefits, and salary at $t-1$.

In a DPRM (and PRM), the inference problem involves computing a marginal probability or a MAP inference over an induced ground Bayesian Network containing random variables represented as nodes [21]. The complexity of inference lies in the number of random variables and its values. The inference problem is proved to be intractable and hard to approximate [21]. Sampling techniques, such as Gibbs sampling and particle filtering, are used for approximate inference in DPRMs and PRMs [21]. Structural learning discovers the parent-child relationships, and quantifies each valid structure according to a scoring function. Parameter learning computes the CPDs given a structure. Maximum likelihood techniques are often used for parameter learning, which identify parameters that maximize a likelihood function [21].

B. A Spatio-Temporal Probabilistic Model

Knowledge of a cell value alone is not sufficient to determine whether it is up-to-date, e.g., knowing a person's street address does not indicate whether it is the *current* address. However, if we learn that a change to a person's residential city very likely triggers a change to their street address (where likelihood is quantified using probabilities satisfying a given threshold), then an update to city followed by the absence of an update to street, indicates that street is likely stale.

To capture this intuition, we extend DPRMs to define a *Spatio-Temporal Probabilistic Model* (STPM) that specifies the joint probability distribution of updates over a database at different times. We differentiate correlated updates (modeled via STPM) from correlated values (modeled via DPRM). For example, an update dependency may exist between attributes MaritalStatus and Surname reflecting the practice of married persons updating their surnames, but there is no dependency between values *married* and their surname values.

Modeling update events requires capturing the spatial and temporal update patterns to provide clues to the types of update relationships that may exist (e.g., causal or co-occurrence). Intuitively, we estimate the currency of a cell by learning from its spatio-temporal update patterns and compute the probabilistic likelihood that c has a missing update.

An *update pattern* is a set of cells with probabilistic dependencies among their updates. In Section IV-B, we define a set of update patterns that capture causal and co-occurrence update relationships. An *update chain* refers to a sequence of causal relationships where an update on one cell triggers a series of subsequent updates on dependent cells. We assume an update pattern is connected via *relational links*, i.e. a sequence of foreign-key references among the participating tuples.

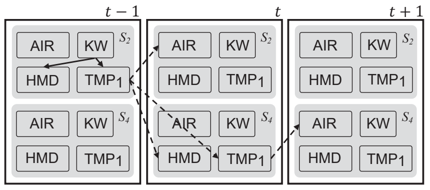


Fig. 4: Spatio-temporal update chain.

Definition 1: An update event instance I_D^t models updates for each cell $c \in D$ at time t. For each c, there is a cell $e^t \in I_D^t$ with value true if c was updated at time t; false if there is no update; or null if it is unknown. The STPM is a DPRM with: (i) an initial PRM $\mathcal{M}_0$ defining the joint probability of update events in an initial instance I_D^0; and (ii) 2TPRM $\mathcal{M}^{\rightarrow}$ defining the transition probability $P(I_D^t|I_D^{t-1})$.

The STPM allows us to infer updates that span over multiple, consecutive time units, forming an update chain. Update chains allow inference over longer temporal update sequences, and broader spatial update patterns (beyond a single tuple) to tuples within a relation, and across relations. For example, Figure 4 shows an update chain over three time units for servers S_2 and S_4 from Table I. An update on S_2 KW at $t-1$ triggers updates to its own HMD and TMP$_1$ (shown as solid arrows), and propagates updates to adjacent server S_4 at times t and $t+1$ (linked via a foreign key, shown as dashed arrows).

C. Estimating Data Currency

We define a STPM model called the UModel that learns from *update* events to estimate the currency of cells in D. We extract the update event instances $I_D^0, ..., I_D^T$, where in each instance, an update event is an update query over the cells in D. We define δ to represent the size of a time unit. If the update frequency for a dataset is known, we set δ equal to this frequency. Otherwise, we assume that the inter-arrival time of updates follows a *Poisson Process* with mean λ [22]. The Poisson distribution is commonly used to model arrival times in stochastic systems. We estimate an initial λ by sampling the inter-arrival times of updates and computing the mean. We prune all intervals that do not fall within a 95% confidence interval of the mean, and re-compute λ with the outliers removed. We set δ equal to the newly computed λ. Our empirical evaluation over real data shows that by using this δ, we achieve an F1-accuracy that is within 3% of the optimal F1 (further details available in [23]).

Learning STPM. The structure of the STPM model is defined via a set of templates, which define the relationships among the random variables of the model (cf. Section IV-B). We learn the model parameters by learning the weights of these templates. In our implementation, we define templates via a set of rules and learn the weights of these rules. The update event instances $I_D^t, t \in [0, T]$ are training data to learn the initial probability distribution $\mathcal{M}_0$, and the transition distribution $\mathcal{M}^{\rightarrow}$. We select any instance I_D^t as a baseline for training $\mathcal{M}_0$, whereas, training $\mathcal{M}^{\rightarrow}$ requires two consecutive instances (I_D^{t-1}, I_D^t).

Currency Inference. Let t_c be the time unit containing the last update query on cell c. Intuitively, c is up-to-date at the current time T if there are no missing updates in the interval $[t_c, T]$. In other words, if we expected an update to c during $[t_c, T]$, and the update is missing from H, then c is stale. Let $current(c)$ represent the probability that c is up-to-date. We estimate $current(c)$ as the marginal probability of all random variables $\{E_c^{t_c+1}, ..., E_c^T\}$ being false, conditional on updates in the parent nodes. $E_c^{t_c+i}$ is an update random variable in the UModel that is true if c was updated at $t_c + i$ (the i-th time unit after t_c). We assume a user-given currency threshold β that is used to select stale cells in $\mathcal{E}$ with $current(c) < \beta$.

$$current(c) = P(E_c^{t_c+1} = \mathsf{false}, ..., E_c^T = \mathsf{false} \mid Pa(E_c^{t_c+1}), ..., Pa(E_c^T)). \quad (1)$$

IV. Probabilistic Currency Inference

We define a set of learning and inference rules to estimate data currency. The premise of these rules is based on update patterns that CurrentClean learns from the update history. We begin with an overview of the DeepDive inference engine.

A. DeepDive and DDlog Rules

CurrentClean uses DeepDive, a statistical learning and probabilistic inference engine [20] that extends probabilistic inference with logical reasoning. DeepDive allows users to define graphical models using probabilistic logical sentences, called *rules*. For example, we can write a rule: *"It is likely that an employee receives a raise, if her coworker received a raise"* using its declarative DDlog language. DDlog is similar to Datalog by defining new *intentional relations* via *extensional relations* based on *extensional data*. DDlog extends the semantics of Datalog with probabilities by assigning weights to some of its rules. A DDlog program is a collection of DDlog rules that defines a probabilistic model called a *factor graph*, where inferred facts are probabilistic, and modeled as ground atoms of the intentional predicates.

A *factor graph* is a graphical probabilistic model defined by a hypergraph with a set of random variables $\mathbf{X}$ as nodes, and a set of hyperedges $\mathbf{F}$ called *factors*. Each factor $\psi \in \mathbf{F}$ ($\psi \subseteq \mathbf{X}$) is associated with a function f_ψ and a weight w_ψ. The function f_ψ assigns a value in $\{-1, 1\}$ to each combination of random variables in ψ. Hyperedges $\mathbf{F}$, functions f_ψ, and weights w_ψ define a factorization of the probability distribution $P(\mathbf{X})$.

A DDlog program may contain three types of rules:

Derivation Rules: Data derivation rules are standard Datalog rules without weights that state how new (intensional) relations are derived from existing (extensional) relations. For example, intensional predicate Coworker in the *head* of the rule, is defined via extensional predicate Employee in the *body* that finds employees working in the same department d: $\mathsf{Coworker(e_1, e_2, d)}$:- $\mathsf{Employee(e_1, d), Employee(e_2, d)}$.

Inference Rules: An inference rule is a DDlog rule with probabilistic intentional predicates, called *variable predicates* (denoted with a ? in the declaration). The *grounding* process instantiates the template rule with input and derived data, and atoms in the predicates define random variables. For example, (2) defines a variable predicate Raise? modeling the likelihood an employee receives a raise if her coworker received a raise.

$$\mathsf{Raise(e_2)} \text{ :- } \mathsf{Coworker(e_1, e_2, d), Raise(e_1)} \quad [@\mathsf{weight(w)}]. \quad (2)$$

The head atoms in inference rules can be replaced with simple logical formulas. Rule (2) can be defined in DDlog by Rule (3) with an implication in its head:

$$\mathsf{Raise(e_1)} \Rightarrow \mathsf{Raise(e_2)} \text{ :- } \mathsf{Coworker(e_1, e_2, d)}, [@\mathsf{weight(w)}]. \quad (3)$$

Each inference rule has a *weight* that indicates the strength of the rule, and the weights of the factors in the factor graph. Weights may be defined as: (i) a given constant; (ii) a fixed value to be learned, represented via a variable (as in (2)); or (iii) a function f of variables in the inference rule, e.g. @weight(f(d)) varies based on the value of d.

Learning Rules: Learning rules provide the declarations to generate training data for learning weights. For example, weights are learned via the extensional predicate SalaryUpdate(e, s) that determine whether employee e receives a raise based on changes to her salary s.

$$\mathsf{Raise(e) = true} \text{ :- } \mathsf{SalaryUpdate(e, s), s > 0}.$$

B. Translating Updates to Inference Rules

We translate the updates in UModel into a DDlog program. We first define the predicates, followed by the rules that define the types of update patterns CurrentClean captures.

Predicates. Predicates define relations that represent database D, query history H, and relationships between cells:

- Cell(c, tid, aid): cell c in tuple tid in attribute aid.
- DomValue(aid, v): value v is in the domain of aid.
- Value(c, v): cell c contains value v in database D.
- Update(c, t, v): c is updated to v at time t w.r.t. H.
- FKey(fkid, kid): foreign key fkid refers to key attribute kid.
- ExpUpdate?(c, t): a variable predicate representing boolean random variables E_c^t in UModel modeling whether c is updated at time t.
- Currency?(c): a variable predicate representing the currency estimation of cell c according to (1).
- LastUpdate(c, t): an intensional predicate stating cell c was last updated at time t.
- Linked($\mathsf{c_1, c_2}$, k): an intensional predicate defining the *relational link* between cells $\mathsf{c_1}$, $\mathsf{c_2}$ with k foreign key references. If $\mathsf{c_1, c_2}$ are in the same tuple, then $k = 0$.

We define the inputs for the program by translating the given event instances of D and H into the extensional predicates Cell, Value, Update, and FKey. We use these predicates to define the data derivation and inference rules for learning, and we estimate currency by inference over ExpUpdate.

DDlog Rules. We present derivation rules, followed by inference rules defining causality, and co-occurrence relationships. Derivation Rules. The following derivation rules define Linked:

$$\mathsf{Linked(c_1, c_2, 0)} \text{ :- } \mathsf{Cell(c_1, r, \text{-}), Cell(c_2, r, \text{-})}, \quad (4)$$

$$\mathsf{Linked(c_1, c_2, 1)} \text{ :- } \mathsf{Cell(c_1, r_1, \text{-}), Cell(cf, r_1, fk), Cell(c_2, r_2, \text{-})}, \quad (5)$$
$$\mathsf{Cell(ck, r_2, key), FKey(fk, key), Value(ck, v), Value(cf, v)}.$$

Rule (4) defines two cells within a tuple. Rule (5) links cells $\mathsf{c_1}$ and $\mathsf{c_2}$ in tuples $\mathsf{r_1}$ and $\mathsf{r_2}$, respectively, with a link length 1 via a foreign key fk to attribute key. CurrentClean is extensible to consider relational links of any length, albeit with a performance overhead (in Section VII-B, we evaluate this overhead for increasing k). We define $\mathsf{LastUpdate}$ of cell c at t, where MAX is a DeepDive function selecting the max value of t: $\mathsf{LastUpdate(c, MAX[t])}$:$-\mathsf{Update(c,t)}$.

Inference Rules. Our inference rules model causality and co-occurrence relationships among updated cells. The first Rule (6), defines a *positive causality* relationship between $\mathsf{c_1}$ and $\mathsf{c_2}$ via the logical implication in the head, i.e., if $\mathsf{c_1}$ is updated, then $\mathsf{c_2}$ must be updated, in either the same or subsequent time unit. The @$\mathsf{weight}(f_1(\mathsf{a_1, a_2, t_2\text{-}t_1}))$ and the function f_1 are defined according to the instantiation of the rule and the specific values of $\mathsf{a_1}$, $\mathsf{a_2}$, and $\mathsf{t_2\text{-}t_1}$.

$$\begin{aligned}&\mathsf{ExpUpdate(c_1,t_1)} \Rightarrow \mathsf{ExpUpdate(c_2,t_2)} \text{ :- } \mathsf{Cell(c_1,\text{-},a_1)},\\ &\mathsf{Cell(c_2,\text{-},a_2), Linked(c_1,c_2,\text{-}), Time(t_1), Time(t_2)}, \mathsf{t_1 \le t_2 \le t_1{+}1},\\ &\qquad [@\mathsf{weight}(f_1(\mathsf{a_1,a_2,t_2\text{-}t_1}))].\end{aligned} \tag{6}$$

The next Rule (7) defines a *co-occurrence* update pattern where updates to $\mathsf{c_1}$ must co-occur with updates to $\mathsf{c_2}$. Intuitively, '$\mathsf{c_1}$ *is updated if and only if* $\mathsf{c_2}$ *is updated.*'

$$\begin{aligned}&\mathsf{ExpUpdate(c_1,t_1)} \Leftrightarrow \mathsf{ExpUpdate(c_2,t_2)} \text{ :- } \mathsf{Cell(c_1,\text{-},a_1)},\\ &\mathsf{Cell(c_2,\text{-},a_2), Linked(c_1,c_2,\text{-}), Time(t_1), Time(t_2)}, \mathsf{t_1 \le t_2 \le t_1{+}1},\\ &\qquad [@\mathsf{weight}(f_2(\mathsf{a_1,a_2,t_2\text{-}t_1}))].\end{aligned} \tag{7}$$

Rule (8) models *negative causality* when updates should not occur. Intuitively, in Rule (8) when $\mathsf{c_1}$ is updated, then $\mathsf{c_2}$ should not be updated in the same or subsequent time unit. In our data center example, this occurs when air conditioning is activated, and no measurements are expected in subsequent time units to allow airflow and temperatures to stabilize.

$$\begin{aligned}&\mathsf{ExpUpdate(c_1,t_1)} \Rightarrow \mathsf{!ExpUpdate(c_2,t_2)} \text{ :- } \mathsf{Cell(c_1,\text{-},a_1)},\\ &\mathsf{Cell(c_2,\text{-},a_2), Linked(c_1,c_2,\text{-}), Time(t_1), Time(t_2)}, \mathsf{t_1 \le t_2 \le t_1{+}1},\\ &\qquad [@\mathsf{weight}(f_3(\mathsf{a_1,a_2,t_2\text{-}t_1}))].\end{aligned} \tag{8}$$

Our model is extensible to capture more complex patterns involving multiple variables. For example, Rule (9) involves three variable predicates in the head associating updates over cells $\mathsf{c_1}$ and $\mathsf{c_2}$ at time $\mathsf{t_1}$ to updates over $\mathsf{c_3}$ at $\mathsf{t_2}$. The rule body is similar to Rule (8) except over three cells.

$$\begin{aligned}&\mathsf{ExpUpdate(c_1,t_1)} \wedge \mathsf{ExpUpdate(c_2,t_1)} \Rightarrow \mathsf{ExpUpdate(c_3,t_2)} \text{ :- }\\ &\mathsf{Cell(c_1,\text{-},a_1), Cell(c_2,\text{-},a_2), Cell(c_3,\text{-},a_3), Linked(c_1,c_3,\text{-})},\\ &\mathsf{Linked(c_2,c_3,\text{-}), Time(t_1), Time(t_2)}, \mathsf{t_1 \le t_2 \le t_1{+}1},\\ &\qquad [@\mathsf{weight}(f_4(\mathsf{a_1,a_2,a_3,t_2\text{-}t_1}))].\end{aligned} \tag{9}$$

We note that contradictions can occur among these rules. For example, negative causality rules such as (8) may contradict positive causality rules of the form (6). However, the UModel learns from a single training instance, where the weight of a rule is determined based on evidence in the data. Strong evidence that supports one rule will refute any contradictory rule, consequently lowering its weight, and its significance during currency estimation and repair.

Learning Rule. Learning rules provide training data for structural and parameter learning, and generate weights for inference rules. Weights are non-negative values, where zero indicates independence between cells, and larger weights represent stronger dependencies between cells. To improve performance, rules with low weights are pruned prior to inference based on a given threshold τ. The following rule assigns labels to instantiate $\mathsf{ExpUpdate}$: $\mathsf{ExpUpdate(c,t) = true}$:- $\mathsf{Update(c,t,\text{-})}$. According to the rule, if there is an Update for c at time t, then $\mathsf{ExpUpdate(c,t)}$ is true, otherwise it is set to false.

To compute $current(c)$, Rule (10) models the likelihood there are no updates since the last time c was updated (at $\mathsf{t_1}$) until the current time. We use $\mathsf{t_2}$ to range over this time period. In DeepDive, we use the Time and ALL predicates to capture all times $\mathsf{t_2}$ where there is no expected update since $\mathsf{t_1}$.

$$\begin{aligned}&\mathsf{Current(c)} \text{ :- } \mathsf{LastUpdate(c,t_1)},\\ &\qquad \mathsf{ALL[!ExpUpdat(c,t_2), t_1 < t_2, Time(t_2)]}.\end{aligned} \tag{10}$$

DeepDive Features. We chose DeepDive for its declarative DDlog rules that are highly modular and flexible, allowing users to extend and customize these rules. DeepDive's inference and learning engine, *Tuffy*, provides scalable inference using a hybrid technique of in-database grounding and in-memory sampling [20]. Tuffy applies a bottom-up grounding of the DDlog rules that exploits the underlying DBMS, and a sampling technique using parallel in-memory search to find the probability of atoms in the ground rules. While our current implementation uses DeepDive, CurrentClean is amenable to other probabilistic inference engines by translating the DDlog rules to the language used in the new engine.

V. Cleaning Stale Data

We introduce a probabilistic repair model called the RModel, and present two repair types for data currency. We then describe how the RModel is instantiated into a DDlog program and present CurrentClean's data repair algorithm.

A. Spatio-temporal Data Repairs

Unlike previous techniques that learn from a static instance of D, CurrentClean uses multiple temporal instances to infer data repairs. By learning the value correlations *over time*, we propose data repairs that consider knowledge of past cell values. CurrentClean computes the most likely repair for a cell c based on the *evolution* of values in c and in its neighbors.

The RModel captures the joint probability distribution of database instances D^t for $t \in [0, T]$. We populate each instance D^t based on the queries in H, i.e., for each cell $c^t \in D^t$, if we observe a value v_c^t at time t, then we assign v_c^t to c^t, otherwise, we assign the value null. The initial PRM is learned from a single instance D^t, and the transition 2TPRM is learned from two adjacent instances D^t, D^{t+1}. Let V_c^t be a random variable denoting the value of cell c at time t. The RModel performs inference over V_c^t according to the type of repair. We consider two types of repairs, modeling the properties of confidence and cardinal-minimality [2].

Most-Likely Repairs (MLR). The *most-likely repair* for a stale cell $c_{err} \in \mathcal{E}$ at time t is the value v that maximizes the following probability. We use MAP inference over $V_{c_{err}}^T$ given the parent random variables $Pa(V_{c_{err}}^T)$. For c_{err} in attribute A, the repair value v is chosen from the domain of A as follows.

$$\arg\max_{v \in Dom(A)} (P(V_{c_{err}}^T = v | Pa(V_{c_{err}}^T))).$$

For example, the most-likely repair for S_2 HMD in Table I is dependent on its TMP_1 and KW values.

Bounded-Cost Repairs (BCR). Complementary to existing minimal-change and cardinality-minimal techniques [2], we propose *bounded-cost repairs* that restrict the magnitude of change to c_{err} according to a distance metric d such that $d(v, v_{c_{err}}) \leq \eta$ where $v_{c_{err}}$ is the stale value in c:

$$\arg \max_{v \in Dom_{\eta, v_{c_{err}}}(A)} (P(V^T_{c_{err}} = v | Pa(V^T_{c_{err}}))).$$

$Dom_{\eta, v_{c_{err}}}(A)$ is a subset of values in $Dom(A)$ within a distance of η from $v_{c_{err}}$. For example, if $\eta = 10$, a most-likely repair of S_2 HMD to 58% is rejected since it lies outside the permitted range of 40 ± 10. The next most-likely value within η is selected as the repair value.

B. Translating Values to Inference Rules

To instantiate the RModel, we define DDlog inference rules to infer repairs. We use the predicates defined in Section IV-B and define a variable predicate $\mathsf{ExpValue?(c, t, v)}$ that specifies the query random variables V^t_c.

$$\mathsf{ExpValue(c_1, t_1, v_1)} \Longleftrightarrow \mathsf{ExpValue(c_2, t_2, v_2)}\ \text{:-}\ \mathsf{Cell(c_1, \text{-}, a_1)}, \quad (11)$$
$$\mathsf{Cell(c_2, \text{-}, a_2), Linked(c_1, c_2, \text{-}), Time(t_1), Time(t_2)}, \mathsf{t_1} \leq \mathsf{t_2} \leq \mathsf{t_1{+}1},$$
$$\mathsf{DomValue(a_1, v_1), DomValue(a_2, v_2)}, \quad [\mathsf{@weight}(f_5(\mathsf{a_1, a_2, t_2\text{-}t_1}))].$$

Rule (11) captures an update pattern in which $\mathsf{c_1}$ and $\mathsf{c_2}$ co-occur at $\mathsf{t_1}$ and $\mathsf{t_2}$, respectively. To provide the input training data, the following learning rule labels the random variables in ExpValue according to the values found in the extensional Update predicate: $\mathsf{ExpValue(c, t, v) = true\ \text{:-}\ Update(c, t, v)}$.

```
Algorithm 1: $CurrentClean(D, H)$
  Input : Database $D$ and history $H$
1 UModel $\leftarrow buildCurrencyModel(D, H)$;
2 $\mathcal{E} \leftarrow \emptyset$; $D^C \leftarrow D$;
3 for $c \in D^C$ do
4     if UModel.$current(c) < \beta$ then $\mathcal{E} \leftarrow \mathcal{E} \cup \{c\}$;
5 RModel$\leftarrow buildRepairModel(D, H)$;
6 for $c_{err} \in \mathcal{E}$ do  $c_{err}$.value $\leftarrow$RModel.$repair(c_{err})$ ;
7 return $D^C$;
```

Together, the above rules define a DDlog program that computes the most-likely values for a cell c at time t, i.e., the value v that maximizes the probability ExpValue(c, t, v). We give details of CurrentClean's repair algorithm in Algorithm 1.

C. Complexity Analysis

CurrentClean's runtime is dominated by the inference complexity over the UModel and the RModel, which is done over ground instances of factor graphs. This inference complexity is NP-hard w.r.t. the number of random variables n_v, which is on the order of $|D| \times |H|$ [21]. The intractability is due to the the large domain sets of the random variables, and the complex relationships among these values (influenced by k).

Given this intractability, we simplify the relationships to reduce the complexity to $n_v \times \log(n_v)$: (i) we restrict the types of update relationships to positive/negative causality, and co-occurrences; (ii) the DDlog rules contain up to three (random) variable predicates; (iii) limiting the relational link length k simplifies the relationships among random variables; and (iv) in the next section, we present a baseline optimization that allows us to re-write implication rules to contain a single variable predicate without sacrificing correctness. We present a set of optimizations to further reduce the runtime by decreasing the number of random variables, n_v, and their domain sets. Lastly, we note that inference over the UModel involves reasoning over binary values (true and false), making it faster than the RModel, which has a larger (albeit finite) attribute domain. The more costly RModel inference is then applied only over the stale cells identified by the UModel.

VI. Optimizing CurrentClean

To scale CurrentClean, we describe two rule modifications as part of our baseline model, and then propose three optimizations to improve CurrentClean's performance. Our experiments show that these optimizations improve runtime by 4.7x.

The Baseline: To reduce the inference complexity, we optimize the inference rules in two ways. First, we limit the number of variable predicates in each rule to two variables, representing binary relations between random variables. Our evaluation shows that binary relationships dominate among high-weight update patterns. We found 89% of the 2-variable rules are in the top 20% of update patterns (ranked by weight), in contrast, only 12% of 3-variable rules are in this set. Second, we replace query random variables with evidence variables to reduce the space of candidate values. For example, in rule (6), we replace $\mathsf{ExpUpdate(c_1, t_1)}$ with $\mathsf{Update(c_1, t_1)}$, and move the predicate to the body. This allows our model to compute $\mathsf{ExpUpdate(c_2, t_2)}$ directly from the history. The resulting rules contain one variable predicate that allows DeepDive to perform efficient Gibbs sampling [20], and reduces the inference complexity from NP-hard to $n_v \times \log(n_v)$. Our evaluation show that with these re-writings, we reduce the enumeration space and incur an average 2.5% loss in F1 accuracy. By default, we include these rule enhancements in CurrentClean, and refer to this as our baseline algorithm.

Opt-1: Pruning Weakly-Correlated Attributes. During inference, we prune weakly-correlated attribute pairs from the enumeration space. We pre-process H to compute the update frequency between every pair of attributes $A_i, A_j \in \mathbf{R}$. We prune pairs $(A_i,\ A_j)$ that do not satisfy the statistical χ^2-test at a given p-confidence level. This reduces instantiations of random variables, and reduces n_v if all instances for an attribute are removed. In our evaluation, we use $p = 95\%$ confidence, and approximately one-third of the attribute pairs are pruned. These attribute pairs represent low-weight update patterns in DeepDive. In Section VII-C, we show that Opt-1 yields a 1.6x runtime improvement over the baseline, and a 1% loss in F1 accuracy. We define a predicate $\mathsf{Irrelevant}(A_i, A_j)$ to model and prune weakly-correlated attribute values in our DDlog inference rules.

Opt-2: Partitioning the Attribute Domain. We reduce the space of attribute value combinations during inference by stratifying values in attributes with large domains or containing continuous values (e.g., attributes such as temperature). By decreasing the attribute(s) domain, we directly reduce the domain of each random variable, simplifying the RModel factor graph. We partition the attribute domain into B buckets using the histogram construction algorithm by Jagadish et. al., [24]. This dynamic programming partitioning (DPP) algorithm computes optimal bucket boundaries for a set of error metrics in quadratic time to the attribute domain size, and linear in the number of buckets. Given an error bound ϵ, the DPP algorithm partitions the data into a minimal number of buckets B with error at most ϵ. We use the *Sum of Squared Error (SSE)* to measure the residual error between each original observation x_i and the bucket approximation h_j for bucket j. One of the natural choices for the bucket approximation is to choose h_j equal to the average of values in bucket j. We set ϵ to be an upper bound on the SSE, and use the *Coefficient of Determination*, $R^2 = 1 - \frac{SSE}{SST}$, $0 \leq R^2 \leq 1$ to compute the SSE value. R^2 measures the amount of variance in the approximation explained by the observations. Intuitively, R^2 measures the goodness of fit of the approximation model to the observations, where values close to 1 indicate a closer fit. The *Sum of Squares Total (SST)* captures the error between the observations and the overall mean. Our evaluation shows that Opt-2 improves the RModel performance by 10-22% with an average 1.5% F1 loss for R^2 ranging from 95-80%.

Opt-3: Reducing the Size of H. Given the potentially large size of H, we prune updates that occur seldomly between attributes, as they do not contribute towards (high-weighted) update patterns. By reducing $|H|$, this reduces n_v in the factor graphs, improving inference runtime. We expand the statistics from Opt-1 to include the time of each update. We model correlated updates using a *dependency cube (d-cube)*, which has been used in temporal FD mining to identify value implication [25]. We define a d-cube over three dimensions: attributes A_i and A_j, and time intervals t. Figure 5 shows a d-cube where a cell is referenced by a triple (A_i, A_j, t) returning the number of updates between attributes A_i and A_j at time t. The time intervals along the z-axis encapsulate the time period covered by H. While the number of time intervals can be large, the number of attributes is fixed, and updates occur at some distribution over t. These factors make the data in the cube very sparse and manageable in practice. For a reference pair (A_i, A_j), the sequence of updates over time defines a *stripe*, as shown in Figure 5 containing cell n_{ij}. For a time t, to quantify the association (A_i, A_j) based on their update frequency, we use the *Normalized Pointwise Mutual Information* (NPMI) measure defined as [24]:

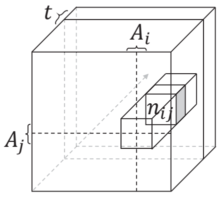


Fig. 5: d-cube.

$$i(A_i, A_j) = ln(\frac{P(A_i, A_j)}{P(A_i) \times P(A_j)})/ - ln\ P(A_i, A_j),$$

where $P(A_i, A_j)$ is the joint update probability of (A_i, A_j), and $P(A_j)$ is the marginal update probability of A_j over the stripe. The pointwise mutual information quantifies the discrepancy between the probability of their coincidence given their joint distribution and their individual distributions, assuming independence. The NPMI ranges from -1 to 1, where $i(A_i, A_j) = -1$ indicates no co-updates, 0 indicates independence, and 1 indicates (A_i, A_j) are always updated together. For a given threshold α, we prune updates (A_i, A_j) where $i(A_i, A_j) < \alpha$. Our evaluation shows that for $\alpha = 0.3$, Opt-3 improves runtime by 70% with a 1.8% accuracy loss.

TABLE II: Real data characteristics.

	Mimic	**Sensor**	**Trans**	**NBA**
N	60,000	58	256,000	3,000
n	27	4	6	44
$\|D\|$	1,620,000	232	1,280,000	1,320,000
$\|H\|$	2,438,327	117,323	5,035,648	2,074,356

VII. Experiments

We evaluate CurrentClean using four real datasets, and compare against state-of-the-art data repair techniques. Our objectives are: (1) We evaluate the accuracy of the UModel and the presence of our update patterns in real data; (2) We study the quality to performance trade-off of our optimizations; (3) We evaluate the comparative accuracy of the RModel to recommend the correct repairs; and (4) We evaluate CurrentClean's scalability and comparative performance.

A. Experimental Setup

CurrentClean is implemented with Java v1.8 using a cluster of Intel Xeon CPUs E5-2687W v4 3.00GHz with 32GB of memory. We use DeepDive v0.8.0 with Postgres 9.2.23. Reported runtimes are the average over three executions.

Datasets. We use four real data collections. Table II gives the data characteristics, showing a range of data sizes w.r.t. the number of entities (N), number of attributes n, number of cells ($|D|$), and number of updates ($|H|$).

Mimic [26]: The Mimic database contains hospital data describing patients vital signs, lab tests, and medications. For example, Arterial PH (APH), #blood tests (BTAN), hemoglobin (Hb), heart rate (HR), monocyte (MONO), red blood cells count (RBC), respiratory rate (RR), systolic blood pressure (SBP), oxygen saturation (SpO2), body temperature (TMP), and white blood cell count (WBC). We extract the update history for one week by comparing successive data snapshots, where the update frequency ranges from 1 min to 1 day. We found 14% of the values to be stale. We evaluate CurrentClean's error detection, repair accuracy, and performance using this dataset.

Sensor [27]: We collected sensor readings from a corporate data center reporting air pressure (AP), humidity (HMD), temperature (TMP), and voltage (V) (every 20s) on server racks over a week. We found 10% of the data values to be stale.

Retail Transactions (Trans) [9]: This dataset contains customer product purchases for a UK retailer from 2010 to 2011. The dataset describes 4207 products, their quantity, price, and purchase date by 4373 customers. To identify co-updates among product purchases, we transform the data such that each record represents all pairs of products purchased in a single transaction by a customer, totaling 256,000 records. We evaluate the prevalence of update patterns and relational chains using this data. A product and customer are considered stale, if there has been no purchase activity for 3 and 6 months, respectively. We found 40% of this data to be stale.

NBA [28]: This dataset provides player statistics for 3000 NBA players, and their teams from 1978 to 2016. We extract the updates each year by comparing successive instances, to reflect salary changes and trades. We use this dataset to evaluate the prevalence of update patterns and relational chains.

Computing the Ground Truth. For each dataset, we consult with domain experts who provided expected update dependencies among the attributes. For example, in the Mimic data: (i) changes in HR trigger SBP change; (2) RR and SpO2 are often co-updated; (3) TMP does not change when RBC changes. We are also provided with a set of ground truth attribute domain values, e.g., 36.5°C $\leq$ TMP $\leq$ 37.5°C, and SpO2 $\geq$ 94%. The full set of attributes update dependencies and value ranges can be found in [23]. To compute a ground truth instance for comparison, we mark all cells in D that do not satisfy the given update dependencies as errors (stale values). Similarly, a proposed repair is considered correct if its value is within the given attribute domain range for a cell.

Comparative Techniques.

Currency Constraints (CC) [16]: This model introduces currency constraints which rely on the data semantics to induce a partial ordering among the tuples. Given a relation, duplicate tuples for an entity are identified and ordered according to the currency constraints, e.g., tuple r_1[Status] = *retired* is more recent than r_2[Status] = *employed* for the same person. We define a set of currency constraints based on temporal relations from domain experts (we refer the reader to [23] due to limited space). For example, for tuples r_1[BTAN] and r_2[BTAN] describing #blood tests for a patient in the Mimic data, if r_1[BTAN] $> r_2$[BTAN], then vital readings such as Hb (hemoglobin) in tuple r_1 are more recent than in r_2, as BTAN values increase over time.

HoloClean [6]: HoloClean provides holistic data repair by combining multiple input signals (integrity constraints, external dictionaries, and statistical profiling), and uses probabilistic inference (DeepDive) to infer dirty values on the current data instance [20]. We input to HoloClean: (i) a set of denial constraints translated from the given update dependencies [23]; (ii) external reference sources such the National Drug Code Directory [29]; and (iii) we profile the data to obtain statistical frequency distributions of each attribute's domain.

ERACER [14]: ERACER models attribute dependencies using a probabilistic graphical model. The model assumes the structure template among attributes is given, and imputes missing values (in the form of nulls) by learning the CPDs. We provide ERACER with 11 attribute dependency templates (listed in [23]) that are derived from the given update dependencies, and we compute baseline CPDs by aggregating over known instances in the data. In our comparative experiments, we set stale values to null, thus treating them like missing data.

IMR [12]: Given a sample of labeled values, the *Iterative Minimum Repair* (IMR) framework repairs time-series data by recommending updates that differ minimally from the labeled truth. High confidence repairs from past iterations are used to determine subsequent repairs [12]. IMR repairs time-series data for a single attribute, and unlike the above baseline techniques, focuses on temporal (not spatial) correlations using historical data. We set IMR parameters as follows: order $p = 3$, convergence threshold τ ranges from 1% to 17%, #maxIterations = 10,000. The label rate γ indicates the percentage of true values. In our experiments, we set $\gamma = (1-e)$ for a given error rate e, and label cells using the ground truth domain.

TABLE III: Parameter values (defaults in bold).

Sym.	Description	Values
β	currency threshold	0.2, 0.4, **0.6**, 0.8, 1
e	error rate (%)	2, **4**, 6, 8, 10
δ	time unit	**69**s (Mimic); **20**s (Sensor) **22**min (Trans) ; **1**yr (NBA)
k	link length	0, **1**, 2, 3, 4
$\lvert H\rvert$	history (%)	**32** (Mimic); **40** (Sensor) **35** (Trans); **24** (NBA)

Parameters. Unless otherwise stated, Table III shows the range of parameter values we use, with default values in bold. We set the time unit, δ, to the update frequency of each dataset, if known. Otherwise, we set $\delta = \lambda$, where λ is the Poisson distribution mean as described in Section III-C. The user-defined currency threshold β ranges in value from 0 to 1, where higher values impose more stringent conditions for satisfying currency. The history size $|H|$ is computed as the percentage of H remaining after applying all optimizations.

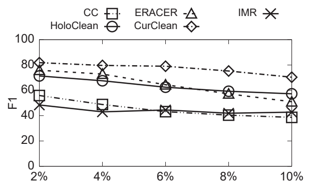
B. Identifying Stale Cells

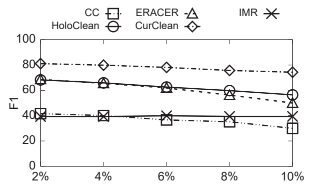
We evaluate the prevalence of stale values in real data.

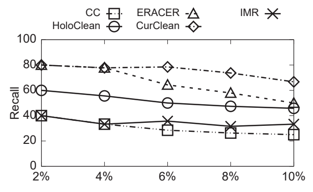
Exp-1: Stale Value Detection Accuracy. We evaluate CurrentClean's error detection accuracy (to detect stale values) against four existing techniques. We identify all errors using the ground truth, and control the error rate e by resolving all errors manually until e% errors remain. Figure 6 shows the F1 and recall scores. CurrentClean achieves an average recall score of 76% and 80% over the Sensor and Mimic datasets, respectively, and 78% F1 over both datasets.

We found CurrentClean learns a richer set of update patterns with longer update chains over the Mimic data. The periodic regularity in the Sensor data helps to boost its precision (and F1-score) to identify stale values. In contrast, the Mimic data contains variable update frequencies, making it more sensitive to δ. That is, if δ is too large, many updates may be unnecessarily aggregated, and the ordering among updates is lost. However, as Figures 6(b) and 6(d) show, our selection method for δ works well to achieve improved accuracy over state-of-the-art techniques.

Comparatively, CurrentClean outperforms HoloClean, CC, ERACER, and IMR in F1 scores by an average +15%, +32%, +15%, and +33%, respectively. HoloClean and CC both use fixed constraints that precisely define the error conditions, but lead to missed stale values. Figures 6(c) and (d) show that CurrentClean outperforms all four methods in recall by


(a) F1 (*Sensor*)


(b) F1 (*Mimic*)


(c) Recall (*Sensor*)

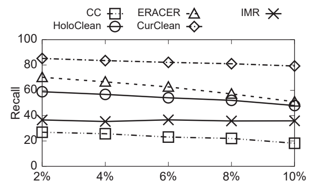

(d) Recall (*Mimic*)

Fig. 6: Error detection accuracy for varying error rate e.

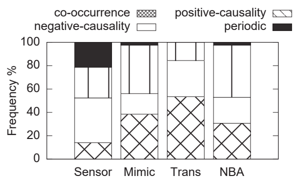


Fig. 7: Pattern freq.

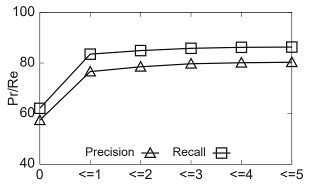


Fig. 8: Accuracy vs. k

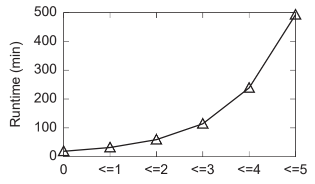


Fig. 9: Runtime vs. k

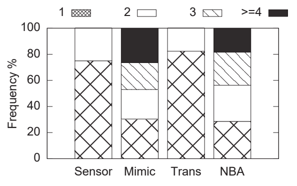


Fig. 10: Upd. chain freq.

an average +34% as many stale values are not captured. ERACER exhibits the greatest sensitivity to increasing e, which negatively influence the learning of the CPDs. IMR is relatively stable across all error rates e, outperforming CC, and achieves improved recall in the Mimic data (vs. Sensor) where larger data fluctuations occur.

Exp-2: Prevalence of Update Patterns. Figure 7 shows the distribution of update patterns for each dataset. As expected, the Sensor data contains the largest proportion of periodic updates (21%), whereas co-occurrence and (positive/negative) causality patterns dominate in the remaining datasets (covering up to 97% of the total updates). The update patterns modeled by CurrentClean occur frequently in practice.

Exp-3: Relational Links: Accuracy and Runtime. We evaluate the influence of relational link length k on the stale detection accuracy using the Mimic data, i.e., how close do updates occur in practice and how do they impact accuracy? Figure 8 shows the precision and recall for patterns with length k. Updates occurring within a tuple (k = 0) lead to 57% and 62%, precision and recall, respectively. By including relational links with $k = 1$, we increase the precision and recall by +19% and +21%, respectively. Further increases for $k \geq 2$, only result in a modest +2% gain. Figure 9 shows the error detection runtime where $k \leq 1$ take 10% of the total runtime, and $k > 1$, consume the remaining 90% of the runtime with only a +3% accuracy improvement. We found that updates within a tuple (k = 0) are by far the most common (occurring 68% of the time), followed by updates with k = 1. Given these results, we henceforth focus on updates involving relational links of length k = 0, 1 for greater efficiency and effectiveness.

Exp-4: Identifying Update Chains. An *update chain* of length l is a sequence of updates over $l + 1$ values. We compute the frequencies of these update chains for varying l as shown in Figure 10. Updates involving two values ($l = 1$) are most common. Datasets with a high reference locality (e.g., Sensor and Trans) exhibit shorter update chains with $l \leq 2$, whereas longer update chains occur (Mimic, NBA) when updates are not as localized, containing 5+ values.

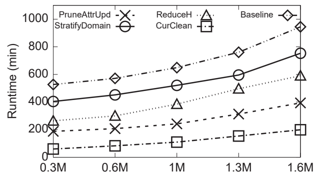
C. Quality to Performance Tradeoff

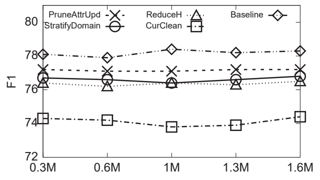
Exp-5: Effectiveness of Optimizations. We evaluate the runtime and accuracy impact of each optimization against the CurrentClean baseline using the Mimic data. For clarity, we refer to Opt-1, Opt-2, Opt-3, as PruneAttrUpd, StratifyDomain, and ReduceH, respectively. We refer to the fully optimized CurrentClean (with all three optimizations) as CurClean. Figures 11(a)-11(d) show the runtimes and F1 performance as we scale $|D|$ and e. Individually, PruneAttrUpd and ReduceH yield the largest performance gains, 1.6x and 70% faster, respectively, than the baseline, and -1% and -1.8% in F1 accuracy. CurClean is 4.7x faster than the baseline with an average 4% F1 accuracy loss. The F1 score degrades rapidly for increasing e due to the unreliability of values, and declines in pruning effectiveness. We note that the optimizations do interact, e.g., applying PruneAttrUpd also reduces $|H|$. Hence, CurClean performance is not necessarily equal to the cumulative performance of individual optimizations.

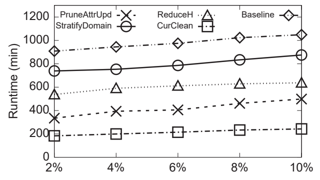
D. Repair Accuracy

Exp-6: Finding the Right Repairs. We compare the accuracy of the most-likely repair (MLR), and bounded-cost repair (BCR) against existing data repair techniques. For BCR, we use Euclidean distance to limit the magnitude of change by setting the distance threshold η to be within 5% of the original value. Figure 12 shows that BCR achieves improved recall in the Mimic data due to the increased number of learned update patterns that cover many stale cells to be repaired.

Figure 12 shows that BCR and MLR achieve superior F1 and recall scores (by +29% and +26%, respectively, over existing techniques), confirming that learning from past value correlations is needed. CC performs poorly, as relying on hard constraints captures on average 30% of the correct repairs (Figure 12(c) and Figure 12(d)). ERACER is sensitive to increasing error rate causing the quality of the learned CPDs to degrade. HoloClean is limited to learning from only the current data snapshot. Lastly, while IMR considers historical data in its likelihood estimation, it does not support spatial


(a) Runtime vs. $|D|$


(b) F1 vs. $|D|$


(c) Runtime vs. e

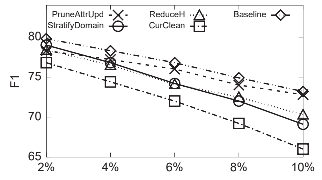
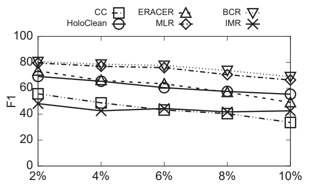
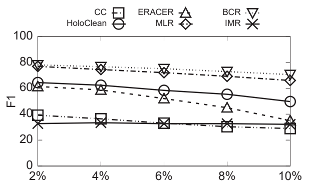
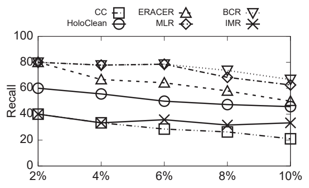

(d) F1 vs. e

Fig. 11: Optimizations: Quality vs. Performance.


(a) F1 (*Sensor*)


(b) F1 (*Mimic*)


(c) Recall (*Sensor*)

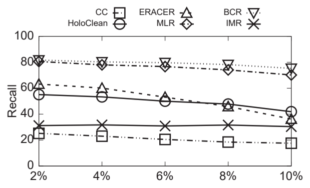

(d) Recall (*Mimic*)

Fig. 12: Data repair accuracy w.r.t error rate e.

correlations among the attribute values, resulting in a different and limited set of repair values.

E. Runtime Performance

Exp-7: Scalability. Figure 13 shows CurrentClean's performance as we scale the number of attributes n using the Mimic data. As expected, the runtimes increase due to increasing $|D|$ and n_v in the factor graphs. To show the impact of our optimizations, the fully optimized CurClean exhibits the smallest rate of increase, providing CurrentClean with improved scalability. Figure 11(a) shows CurrentClean's scaleup w.r.t. $|D|$, and the effectiveness of Opt-1 and Opt-2. We observe similar trends for increasing error rate e (Figure 11(c)), and $|H|$ (due to Opt-3, we refer to [23] for resulting graphs). These results align with CurrentClean's (optimized) complexity of $n_v \times \log(n_v)$. Lastly, as expected, Figure 9 shows the exponential scale-up w.r.t. k, due to the complex relationships among the random variable values.

Exp-8: Comparative Performance. Table IV shows the total wall-clock comparative runtimes. CurrentClean incurs additional time due to the combinatorial explosion of reasoning over the *spatial and temporal* correlations among relational attributes, and their updated values. CC relies on declarative constraints with no probabilistic inference. ERACER and HoloClean reason about errors by considering only value dependencies at a single point in time. IMR repairs non-labeled values within a time period for a single cell; hence, the history is limited to the given cell. Unlike previous techniques that ignore relational links, we exploit these links to identify correlated updates spanning multiple relations.

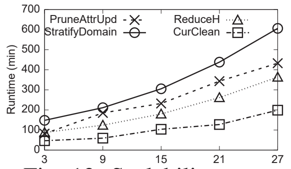


Fig. 13: Scalability w.r.t. n.

TABLE IV: Comparative runtimes.

Dataset	CurClean	HoloClean	CC	ERACER	IMR
Mimic	199m	129m	6m	64m	16m
Sensor	83s	48s	2s	20s	0.2s

TABLE V: Example update patterns.

Update pattern	F1
(i) ΔWBC $\rightarrow$ ΔTMP (+ve)	+2.5
(ii) ΔRBC $\rightarrow$ ΔSpO2 (+ve)	+2.3
(iii) ΔMONO $\leftrightarrow$ ΔWBC (co-occur)	+2.8
(iv) ΔHb $\rightarrow$ $\neg$(ΔTMP) (-ve)	+0.4

Case Study. We assess the impact of our proposed rules by computing the difference in F1 accuracy (over the UModel and RModel) with and without the modeled rule. Positive causality rules showed the greatest benefit contributing 22% and 45% towards the UModel and RModel F1, respectively. In contrast, negative causality rules only contribute 3.5% and 3.8%. Our learned update patterns align with 70% of the ground truth, e.g., updates such as ΔHb$\rightarrow\Delta$RBC contributed 5% and 7% towards the UModel and RModel F1, respectively (hemoglobin carries oxygen in red blood cells).[1]

Table V provides examples of discovered update patterns (that are not part of the ground truth), and their average F1 impact. For example, in pattern (i) as patients experience an infection, their white blood cell count (WBC) increases, along with their body temperature (TMP). Pattern (iv) states that changes in Hb should not trigger changes in TMP, otherwise alerting to a possible abnormality. These new patterns support knowledge discovery and augmentation of the ground truth. Lastly, we found BCR repairs improved F1 repair accuracy by +2.4% for attributes with less than 4% variance in its values (such as SpO2, APH). In contrast, MLR repairs achieved +1.8% F1, for attributes with variance greater than 10%, as bounding the magnitude of change becomes more difficult.

VIII. Related Work

Temporal-based Cleaning. Currency constraints express currency relationships via pre-defined constraints based on the data semantics [16]. Our evaluation has shown that relying on fixed constraints do not capture temporal and spatial dependencies that exist among updates and their values. Recent work

[1] The rule ΔHb$\rightarrow\Delta$RBC is an instance of (6), but in a compact form for brevity. We list only participating attributes, and omit times, tuple ids and values. Positive, negative causality, and co-occurrence rules are specified with $\rightarrow$, [$\rightarrow\neg()$], and $\leftrightarrow$, respectively. Δ represents a change in attribute value.

has explored cleaning time-series data that define *speed constraints* to identify and repair values based on expected rates of change [11], applying minimum changes given labeled ground truth [12], repairing imprecise timestamps [13], and generating repairs according to maximum likelihood estimation [30]. All these techniques clean uni-variate time-series data relative to an expected rate of change based on past time units, which is not the focus of our work. CurrentClean learns spatial and temporal correlations at the schema and data level to propose repairs for stale values.

Temporal Dependencies. Recent approaches have studied temporal locality between data values to mine for temporal FDs over noisy web data, while fixing outliers and inconsistent data values during the mining process [25]. Temporal dependence is used to understand the behaviour of a user cohort to predict their future behaviour [31]. This *recurrent cohort analysis* identifies causative and dependent behaviour. In similar spirit, we study causality and co-occurrence update patterns to understand their influence on data currency, expected updates, and data cleaning. However, their model proposes table transformation and query operators over a database to perform recurrent cohort analysis, which is not the focus of our work.

Probabilistic and Holistic Data Cleaning. As introduced in Section VII-A, ERACER [14] and HoloClean [6] learn from given attribute templates and training samples to infer clean values. Our evaluation has shown that neither approach recognize update patterns across the database cells, and suffer from low recall to capture stale values. The LLUNATIC framework supports more holistic data cleaning by using a general form of equality generating dependencies (EGDs) covering a broad set of dependencies [32]. To repair error cells, LLUNATIC relies on master data as a source of repair values, and does not consider historical relationships between the data values. The SCARED approach learns from clean portions of the data to find bounded repairs according to maximum likelihood [15]. While similar in spirit, these probabilistic techniques differ from our work as they repair values to maximize data accuracy and consistency (rather than currency). Error identification and repair is w.r.t. a fixed time, whereas CurrentClean learns causative and co-occurrence update relationships over time.

IX. Conclusion and Future Work

We present CurrentClean, a probabilistic system for detection and repair of stale values. We argue that currency is a relative notion dependent on an individual entity's update patterns. We propose update and repair models, and instantiate them using inference rules that reason over current and past cell values. Our evaluation shows that CurrentClean learns update patterns that effectively identify and repair stale data. Our next steps include extending H to include incorrect updates, and extending the models to online settings.

References

[1] C. Batini and M. Scannapieco, *Data and Information Quality: Dimensions, Principles, and Techniques*. Springer-Verlag, Inc., 2016.

[2] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," in *SIGMOD*, 2005, pp. 143–154.

[3] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Towards certain fixes with editing rules and master data," *VLDB J.*, vol. 21, no. 2, pp. 213–238, 2012.

[4] Y. Cao, W. Fan, and W. Yu, "Determining the relative accuracy of attributes," in *SIGMOD*, 2013, pp. 565–576.

[5] S. Baskaran, A. Keller, F. Chiang, L. Golab, and J. Szlichta, "Efficient discovery of ontology functional dependencies," in *CIKM*, 2017, pp. 1847–1856.

[6] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "HoloClean: Holistic data repairs with probabilistic inference," *PVLDB*, vol. 10, pp. 1190–1201, 2017.

[7] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "KATARA: A data cleaning system powered by knowledge bases and crowdsourcing," in *SIGMOD*, 2015, pp. 1247–1261.

[8] Australian Govt., "Smart city sensor data," https://data.gov.au, 2017.

[9] D. Chen, "Online retail data set," https://archive.ics.uci.edu/ml/, 2015.

[10] W. Fan and F. Geerts, *Foundations of Data Quality Management*. Morgan & Claypool Publishers, 2012.

[11] S. Song, A. Zhang, J. Wang, and P. S. Yu, "SCREEN: Stream data cleaning under speed constraints," in *SIGMOD*, 2015, pp. 827–841.

[12] A. Zhang, S. Song, J. Wang, and P. S. Yu, "Time series data cleaning: From anomaly detection to anomaly repairing," *PVLDB*, vol. 10, no. 10, pp. 1046–1057, 2017.

[13] H. Zhang, Y. Diao, and N. Immerman, "Recognizing patterns in streams with imprecise timestamps," *PVLDB*, vol. 3, no. 1-2, pp. 244–255, 2010.

[14] C. Mayfield, J. Neville, and S. Prabhakar, "ERACER: a database approach for statistical inference and data cleaning," in *SIGMOD*, 2010, pp. 75–86.

[15] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid, "Don't be SCAREd: Use SCalable Automatic REpairing with maximal likelihood and bounded changes," in *SIGMOD*, 2013, pp. 553–564.

[16] W. Fan, F. Geerts, N. Tang, and W. Yu, "Inferring data currency and consistency for conflict resolution," in *ICDE*, 2013, pp. 470–481.

[17] X. L. Dong, A. Kementsietsidis, and W.-C. Tan, "A time machine for information: Looking back to look forward," *SIGMOD Rec.*, vol. 45, no. 2, pp. 23–32, 2016.

[18] T. Bleifuß, L. Bornemann, T. Johnson, D. V. Kalashnikov, F. Naumann, and D. Srivastava, "Exploring change - A new dimension of data analytics," *PVLDB*, vol. 12, no. 2, pp. 85–98, 2018.

[19] P. J. Denning, "The locality principle," *Communications of the ACM*, vol. 48, no. 7, pp. 19–24, 2005.

[20] J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré, "Incremental knowledge base construction using DeepDive," *PVLDB*, vol. 8, no. 11, pp. 1310–1321, 2015.

[21] D. Koller and N. Friedman, *Probabilistic Graphical Models*. The MIT Press, 2009.

[22] H. M. Taylor and S. Karlin, *An Introduction To Stochastic Modeling*, third edition ed. Academic Press, 1998.

[23] "CurrentClean: Extended evaluation," http://www.cas.mcmaster.ca/~zhengz13/currentclean.html, 2019.

[24] H. V. Jagadish, Koudas, Muthukrishnan, Poosala, and Suel, "Optimal histograms with quality guarantees," in *VLDB*, 1998, pp. 275–286.

[25] Z. Abedjan, C. G. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker, "Temporal rules discovery for web data cleaning," *PVLDB*, vol. 9, no. 4, pp. 336–347, 2015.

[26] A. Johnson, T. Pollard, M. Ghassemi, B. Moody, P. Szolovits, and R. Mark, "MIMIC-III, a freely accessible critical care database," *Scientific Data*, vol. 3, no. 160035, 2016.

[27] Cinnos Mission Critical, "Data center readings," https://www.cas.mcmaster.ca/~zhengz13/Dataset/Sensor.rar, 2018.

[28] J. Grosz and N. Rippner, "NBA player data 1978-2016," https://data.world/jgrosz99/nba-player-data-1978-2016/, 2017.

[29] The Food and Drug Administration (FDA), "National Drug Code Directory," https://www.fda.gov, 2018.

[30] A. Zhang, S. Song, and J. Wang, "Sequential data cleaning: A statistical approach," in *SIGMOD*, 2016, pp. 909–924.

[31] Q. Cai, Z. Xie, M. Zhang, G. Chen, H. V. Jagadish, and B. C. Ooi, "Effective temporal dependence discovery in time series data," *PVLDB*, vol. 11, no. 8, pp. 893–905, 2018.

[32] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "The LLUNATIC data-cleaning framework," *PVLDB*, vol. 6, no. 9, pp. 625–636, 2013.