ENFORCEMENT AND REFINEMENT OF DATA CURRENCY AND DATA CONSISTENCY

ENFORCEMENT AND REFINEMENT OF DATA CURRENCY AND DATA CONSISTENCY

BY

ZHENG ZHENG, M.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

© Copyright by Zheng Zheng, March 2022

All Rights Reserved

Doctor of Philosophy (2022) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE: Enforcement and Refinement of Data Currency and Data Consistency

AUTHOR: Zheng Zheng

M.Sc. (Computer Science),

The University of Chinese Academy of Sciences, Beijing, China

SUPERVISOR: Dr. Fei Chiang

NUMBER OF PAGES: xviii, 170

Abstract

As businesses generate and consume an increasing amount of data, enforcing and maintaining data quality is a critical task for decision making, operational efficiency and accurate data analysis. Recent studies have shown that data scientists spend 60-80% of their time cleaning and organizing data in preparation for data analysis [1]. This effort motivates the need for effective data cleaning tools to reduce the user burden. Data quality is often measured along a set of dimensions: accuracy, completeness, consistency, reliability, and currency. Achieving all these data characteristics is often not possible in practice due to personnel costs, and for performance reasons.

This thesis focuses on tackling three problems in two data quality dimensions: data currency and data consistency. We first address the problem of estimating data currency in a relational database, and we argue that data currency is a relative notion that is dependent on an individual's update pattern. This pattern can have spatial and temporal variance among individuals. By learning the patterns from an update history, we present a probabilistic system for identifying and cleaning stale values. Secondly, we explore the problem of estimating data currency in distributed

database settings. Replicas of the same data item often exhibit varying consistency levels when executing read and write requests due to system availability and network limitations. When one or more replicas respond to a query, estimating the currency of the returned data item is essential for applications requiring timely data. Depending on how confident the estimation is, the query may dynamically decide to return the retrieved replicas, or wait for the remaining replicas to respond. We present a system that accurately estimates whether the retrieved replicas are current or stale, and guarantees that the estimation satisfies a user-given confidence threshold. Using this confidence-bounded replica currency estimation, we implement a novel DYNAMIC consistency level in the open-source, NoSQL database, Cassandra. Finally, we tackle the problem of resolving inconsistencies in a database. Data consistency is often measured by whether the data adheres to a set of data quality rules. Recent work has proposed a new class of data quality rules that considers the data semantics with respect to (*w.r.t.*) an ontology [9]. As the data evolves w.r.t. these rules and ontology, we propose a system to re-align and repair the data and the ontology w.r.t these ontological rules.

Dedicated to my family, my love and my friends.

Acknowledgements

First, I would like to offer my most sincere gratitude to my supervisor, Prof. Fei Chiang. Throughout my Ph.D. study, Prof. Chiang guided me in developing research ideas, writing research papers, and giving academic presentations. I shall be indebted for everything I have learned from her on both academic levels and personal levels. As I pursue my academic career, Fei will always be my advisor, mentor, and dear friend.

I am deeply grateful to my thesis committee members, Prof. Rong Zheng and Prof. Douglas Down, for the discussions throughout my Ph.D. career and insightful comments on this thesis. I would like to thank Prof. Jiannan Wang, for serving as my external reviewer and offering valuable suggestions.

I also would like to thank all the collaborators I have worked with over the years: Prof. Lukasz Golab, Prof. Jaroslaw Szlichta, Prof. Shaoxu Song, Prof. Mostafa Milani, Dr. Yu Huang, Morteza Alipourlangouri, Yu Sun, Tri Minh Quach, Zhi Qu, Longtao Zheng, Ziyi Jin and many others. I would not have had a successful Ph.D. without them.

Finally, this dissertation would not have been possible without my parents and Nancy, for their continuous support and endless love. They are willing to give me everything and expect nothing in return.

Preface

This "sandwich-based" thesis consists of previously published papers over the course of the dissertation. In particular, the work in Chapter 3 was done in collaboration with Y. Sun and S. Song from Tsinghua University.

1. In Section 3.2, Z. Zheng contributed prediction model in idealistic scenario. Y. Sun and S. Song contributed in confidence bounds for replica currency.

2. In Section 3.3, Y. Sun and S. Song contributed in currency estimation and confidence bounds in realistic scenario.

Chapter 3 contains the same content as published in the conference paper [92].

Contents

Abstract **iii**

Acknowledgements **vi**

Preface **viii**

1 Introduction **1**

1.1 Data Currency 3

1.2 Replica Currency Estimation 6

1.3 Data Consistency 9

1.4 Contributions 12

1.5 Thesis Organization 14

2 CurrentClean: Spatio-temporal Cleaning of Stale Data **15**

2.1 Introduction 15

2.2 The CurrentClean Framework 21

2.3 Identifying Stale Data 24

2.4 Probabilistic Currency Inference 30

2.5 Cleaning Stale Data 37

2.6 Optimizing CurrentClean 41

2.7 Experiments 45

2.8 Related Work 58

2.9 Conclusion and Future Work 60

3 Confidence Bounded Replica Currency Estimation **61**

3.1 Introduction 61

3.2 Ideal Scenarios 67

3.3 Realistic Scenarios 76

3.4 Cassandra Implementation 88

3.5 Experiments 91

3.6 Related Work 107

3.7 Conclusions 110

4 OFDClean: Contextual Data Cleaning with Ontology Functional Dependencies 112

4.1 Introduction 113

4.2 Preliminaries and Definitions 117

4.3 Data Cleaning with OFDs 120

4.4 Sense Assignment 126

4.5 Computing Repairs 137

4.6 Experiments 141

4.7 Related Work 149

4.8 Conclusions 151

5 Conclusion and Future Work 152

5.1 Future Work 154

List of Figures

1.1 Update patterns in a data center. 5

1.2 A cluster with replication factor 4, running across multiple data centers. 8

1.3 Ontologies over a *clinical* dataset. 10

2.1 Update patterns 17

2.2 CurrentClean system overview. 22

2.3 (a) PRM CPD (b) 2PRM. 25

2.4 Spatio-temporal update chain. 28

2.5 d-cube. 44

2.6 Error detection accuracy for varying error rate e. 50

2.7 Pattern freq. 53

2.8 Accuracy vs. k 53

2.9 Runtime vs. k 53

2.10 Upd. chain freq. 53

2.11 Error detection accuracy for varying error rate e. 54

2.12 Error detection accuracy for varying error rate e. 55

2.13 Scalability w.r.t. n. 56

3.1 A cluster with replication factor 4 runs across multiple data centers. Consistency indicates that a read returns the most recently written version, i.e., v_{24}, the 4th version stored in replica r_2. Unfortunately, this is not always possible due to system availability and network partitioning. It is highly demanded to estimate whether the response replica, e.g., v_{44}, is current or stale. 62

3.2 Replica currency estimation in an ideal scenario. The update history for all replicas is propagated to a responding node for learning and currency estimation. 68

3.3 Prediction model for consecutive replica versions. 70

3.4 Replica currency estimation with model generalization bound w.r.t. γ in Formula (3.2.5). 76

3.5 Replica currency estimation in a realistic scenario, where versions since the last synchronization are not available nor shared, such as at T_{42}. . 80

3.6 Predicting multiple, unseen replica versions. The predictions (shown as red arrows) are introduced in Formula (3.3.3), and the computed bounds (shown as blue arrows) are guaranteed by Formula (3.3.4). . . 82

3.7 Replica currency estimation with model generalization bound w.r.t. β_q (Formula (3.3.8)), and γ_q (Formula (3.3.9)). 85

3.8 Replica currency estimation by varying training rates over Btcusd data. 94

3.9 Replica currency estimation by varying confidence thresholds η 1 e^{x} over Btcusd data. 95

3.10 Varying time intervals between synchronizations intervals with two consecutive node synchronizations (e.g., T_{42} in Figure 3.5 and the one before it) using Btcusd data. 96

3.11 Varying time intervals between the latest node synchronization and query time (e.g., T_{42} and ζ in Figure 3.5) 96

3.12 Time cost for varying synchronization and query time intervals using Btcusd data 97

3.13 Query loads over (a) network partition with separate reads/writes and (b) node failure with suspended service 98

3.14 Replica currency estimation for varying time lengths of network partition (Figure 3.13(a)) using Btcusd data 98

3.15 Replica currency estimation for varying node failure times (Figure 3.13(b)) using Btcusd data 99

3.16 Varying number of node failures 99

3.17 Query loads with (a) homogeneous and (b) skewed updates and reads 100

3.18 Replica currency estimation by varying the skewness of updates and accesses (Figure 3.17(b)) using Btcusd data 101

3.19 Query load by varying write consistency levels (varying number of write replicas) 102

3.20 Replica currency estimation by varying write consistency levels, according to λ using Btcusd data 102

3.21 DYNAMIC consistency level under various confidence threshold η $1 \quad e^{\ x}$ using Btcusd data 103

3.22 DYNAMIC consistency level under various confidence threshold η $1 \quad e^{\ x}$ using Bike data 105

3.23 DYNAMIC consistency level under various number of servers using Btcusd data 106

4.1 Medical drug ontology. 114

4.2 OFDClean framework. 125

4.3 Sense assignment. 128

4.4 $x_{2\lambda_1}$, $x_{3\lambda_1}$ distributions. 132

4.5 $x_{2\lambda_2}$, $x_{3\lambda_2}$ distributions. 132

4.6 Dependency graph. 133

4.7 Conflict graph for Table 4.2. 140

4.8 Sense assignment effectiveness (Clinical dataset). 144

4.9 OFDClean Performance. 146

4.10 Acc. vs. $inc\%$ 147

4.11 Acc. vs. $|\Sigma|$. 147

List of Tables

1.1 Medical monitoring metrics. 4

1.2 Clinical Dataset 9

2.1 Monitoring metrics. 17

2.2 Real data characteristics. 46

2.3 Parameter values (defaults in bold). 49

2.4 Comparative runtimes. 57

2.5 Example update patterns. 57

3.1 Notations. 67

4.1 Sample clinical trials. 113

4.2 Sample clinical trials data. 140

4.3 Sample repairs. 140

4.4 Parameter values (defaults in bold) 143

4.5 Varying N runtimes. 145

4.6 OFDClean runtimes (m) 148

Chapter 1

Introduction

Data quality is one of the most pressing challenges in data management since real-life data is often dirty: inconsistent, duplicated, inaccurate, incomplete, and out-of-date. Poor quality data across businesses and the government costs the U.S. economy $3.1 trillion a year, according to a report by InsightSquared [28]. A survey conducted by CrowdFlower, shows that data scientists spend 60-80% of their time cleaning and organizing the data [1]. This highlights the need for data quality management, to improve the quality of the data along a set of dimensions including accuracy, completeness, consistency, reliability and currency [11, 38]. *Data accuracy* refers to the closeness of values in a database to the entities that the data represents. *Data completeness* measures the amount of missing information to answer queries. *Data consistency* refers to whether multiple occurrences of a real-world entity or attribute are represented by the same value in the database, and aims to detect inconsistencies and conflicts in the data. *Data currency* identifies the current value

of entities represented by tuples in the database. *Data reliability* measures how much information derives from an authoritative source [11].

Achieving all these data characteristics is a non-trivial problem because of high costs, performance overhead, or interactions between the dimensions. Existing data cleaning techniques often concurrently improve one or two of the above dimensions. Existing work uses master data, user feedback and knowledge bases to improve data reliability [44, 37, 42, 100, 97], maximize accuracy [10, 20, 29] and propose holistic systems to address incompleteness [25, 31, 24, 83]. Quantitative data cleaning approaches involve statistical, probabilistic and machine learning methods to identify abnormal behaviors and errors, and infer the correct values according to expected statistical distributions [14, 32, 82]. Despite a wealth of data cleaning techniques for improving data quality, there are only a few of which address data currency. 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. However, inferring data currency is a difficult problem if the timestamps are missing or imprecise.

Identifying and repairing stale data in distributed database settings is also a critical problem. Replicas of the same data item often exhibit varying consistency levels when executing read and write requests due to system availability and network limitations. When one or more replicas respond to a query, estimating the currency (or staleness) of the returned data item (without accessing the other replicas) is essential for applications requiring timely data. Depending on how confident the estimation is, the query may dynamically decide to return the retrieved replicas, or wait for the

remaining replicas to respond. The replica currency estimation is expected to be accurate and extremely time efficient without introducing large overhead during query processing.

Along with data currency, data consistency is one of the most important properties of data quality to ensure data trustworthiness such that the same value exists across multiple occurrences. Integrity constraints (ICs) are commonly used data quality rules for preserving data consistency. Any part of the data that does not conform to a given set of ICs is considered erroneous, also known as a violation of ICs. Constraint-based cleaning approaches [17, 23, 51, 96] rely on ICs to repair data values such that the data and constraints are consistent.

1.1 Data Currency

Data currency refers to how promptly the data is updated to reflect actual changes in the real world [38], which plays a vital role in traditional business intelligence applications. For example, in healthcare domains, reporting stale data may threaten patient safety. Wenfei Fan et al. [39], propose a type of data quality rules, namely currency constraints, that identify the most recent entity in a database. Currency constraints model currency relationships based on data semantics, which induces a partial ordering among the tuples. For example, a currency constraint states that if an employee's title is "junior" and "senior" in two records t_1 and t_2, respectively, then we expect that t_1 precedes t_2, since an employee must be junior before achieving a senior title. However, currency constraints rely on domain expertise, where manual

specification is needed.

Out-of-date or stale data occurs when the real world changes are not captured by the database (i.e., the updates are missing), which may be caused by human error, system malfunction, etc. Inferring data currency is not a trivial problem if the timestamps are missing or imprecise, since it requires timely knowledge about real world entities, which is represented in data and their last changes. In the real world, missing updates for an entity often depend on the spatial and temporal correlations between updated values, as shown in the following example.

Example 1: Table 1.1 shows patient data monitoring vital signs and blood test results for 4 patients ($p_1 \quad p_4$) in a hospital, which include respiratory rate (RR), the amount of hemoglobin (Hb), oxygen saturation (SpO2), and heart rate (HR). Data currency is important in monitoring a patient's physical condition with these medical indicators so that the doctor can be notified in time when an issue occurs. Figure 1.1 shows the updates history of different medical measurements *w.r.t.* the patients s_2 and s_4 over a period of time, and missing updates are shown as circles. ∎

patient	**RR**	**Hb**	**SpO2**	**HR**
p_1	21	42	none	1
p_2	28	40	high	5
p_3	19	43	none	3
p_4	26	57	none	2

Table 1.1: Medical monitoring metrics.

Spatial correlations. First, we show that data currency may be influenced by spatial updates of neighboring cells within a tuple. In a relational database, a cell

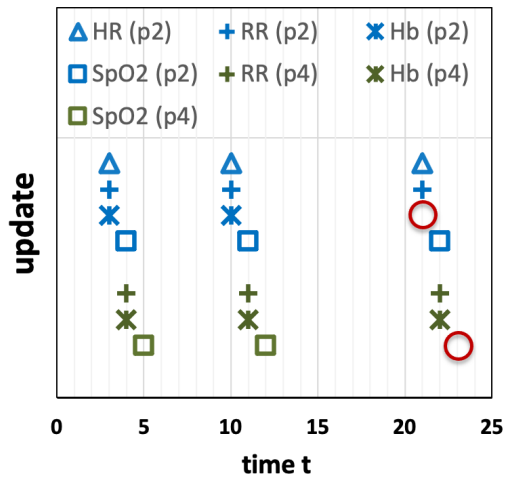


Figure 1.1: Update patterns in a data center.

refers to the unit where a row (tuple) and column intersect. Generally, increased heart rate triggers increases in respiratory rate and hemoglobin. Consider the heart rate changes of patient p_2 (highlighted with a blue triangle) at t $\mathsf{t}3, 10, 21\mathsf{u}$ in Figure 1.1. These changes trigger updates of respiratory rate and hemoglobin of p_2 (highlighted with a blue cross and star, respectively) at the same time, which should be captured by the monitors and recorded in Table 1.1. However, the update for hemoglobin of p_2 is missing at t 21 (highlighted with a red circle), which causes $p_2\mathsf{r}Hb\mathsf{s}$ to be stale at the current time (t 25).

Temporal correlations. Beyond the discussion of spatial correlations within the same time unit, we consider correlations across time. Figure 1.1 shows that increased the respiratory rate (highlighted with a green cross) of p_4 (at time t $\mathsf{t}4, 11\mathsf{u}$) allows more oxygen to get in and combine with hemoglobin, which causes changes in SpO2 of p_4 in the next time unit (t $\mathsf{t}5, 12\mathsf{u}$).

In the above example, we investigate the importance of spatial and temporal relationships among the cells in repairing stale values. We claim that data currency is a relative notion based on individual spatio-temporal update patterns. Having knowledge of a cell's historical update patterns can help determine the currency of the cell. Existing techniques [91, 105] ignore these update patterns which may lead to incorrect or inconsistent repairs. Thus, new techniques are needed to recommend data repairs by considering the spatio-temporal update relationships via past updates.

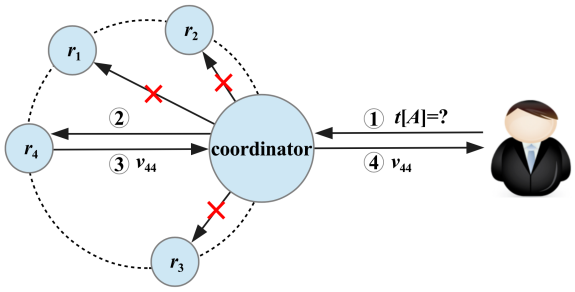
1.2 Replica Currency Estimation

Data currency is often synonymously referred to as data freshness in distributed database settings [38]. Different values of the same entity arise if update queries are not applied simultaneously over all replicas. While each of these values may have been correct at some time, answering queries across these multiple value versions is challenging. Distributed data stores with replication, such as BigTable [21], HBase [50], Dynamo [34], and Cassandra [69], carry different versions of a data item (i.e., a key-value pair, or a cell in a table), where timestamps are used to determine their currency. Replicated storage for large scale applications must consider the trade-off between stronger forms of consistency and performance requirements. On the one hand, stronger consistency provides improved guarantees of data currency. On the other, it often requires heavier-weight implementations that increase latency and/or decrease throughput [71], which leads to limitations in scalability and availability [53], and it is unable to handle network partitions [57].

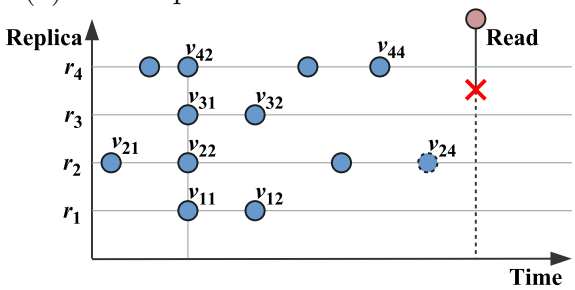
By manually specifying a consistency level, existing systems such as Cassandra wait for a fixed number of replicas (e.g., ONE, QUORUM, ALL) to respond w.r.t. the success of read/write operations. In cases of workload surge, the write consistency level ONE is preferred, i.e., returning to the client as long as one node is written, to ensure high throughput. To achieve strong consistency, where all client reads view the most recent write, the read consistency level ALL is required, which waits for all replicas to respond, given the aforesaid write consistency level ONE. However, this increases latency.

Existing tunable consistency levels sacrifice some consistency to achieve total availability. For instance, in Cassandra, the read consistency level ONE or QUORUM chooses to retrieve, respectively, one or more than half of the replicas. Obviously, the returned replicas may be stale (as illustrated in Figure 1.2), given the aforesaid write consistency level ONE. This solution falls short in two aspects: (1) The client has no information about the currency of the returned values from part of the replicas. (2) The consistency level is tuned w.r.t. the number of responding replicas rather than the likelihood of returning replicas with current values. Without any estimation of data currency, it is difficult for users to configure an appropriate consistency level.

Example 2: Figure 1.2(a) shows a system with replication factor 4. Let v_{ij} denote the j-th version of a cell in the i-th replica. Figure 1.2(b) shows the latest version v_{24} of the cell is stored only in replica r_2, due to write efficiency. Read requests are also processed at different consistency levels, again for either read efficiency or network failure. With read consistency level ONE, Cassandra immediately returns the version held by the first node that responds to the query [69], i.e., v_{44} of replica



(a) Four replicas in a cluster of nodes.



(b) Versions v_{ij} of a cell written to four replicas at different times.

Figure 1.2: A cluster with replication factor 4, running across multiple data centers.

r_4. We can see that version v_{44} is stale, given the more recent v_{24}. However, without further accessing the other replicas, r_1 r_3, the system cannot determine whether the retrieved version v_{44} is current or stale. ❙

At consistency level ONE, as illustrated in Figure 1.2, the returned replica may be stale. Alternatively, if we adopt the consistency level ALL, waiting for all replicas to respond, it blocks the read request (given the network partitioning in the cluster). The QUORUM level achieves a compromise by retrieving more than half of the replicas. However, the aforesaid issues regarding waiting for a fixed number of replicas, unfortunately, still exist.

1.3 Data Consistency

Data consistency is an important dimension to preserve data quality, which refers to the coherence of a data item and often uses integrity constraints (ICs) to specify data quality requirements. ICs are a set of rules that provide a mechanism for ensuring that data conforms to guidelines specified by experts. Data that are inconsistent w.r.t the ICs are identified as erroneous, and updates to the data are generated to re-align the data with the ICs. Functional dependencies (FDs) are one of the most widely used ICs. For example, consider an FD F_1 : r*Symptom*s Ñ r*Medicine*s over Table 1.2, which states that any two tuples with the same value on attribute *Symptom* should have the same value on attribute *Medicine*. Then tuples t_1 t_2 violate F_1 since t_1r*Medicine*s t_2r*Medicine*s. The bulk of existing work has focused on using functional dependencies (FDs) to define the attribute relationships that the data must satisfy [17, 82]. However, these approaches use the ICs that rely on syntactic definitions of equality without capturing the data semantics. For example, with domain knowledge from a clinical ontology (Figure 1.3(b)), we see that the values participate in an inheritance relationship, which states 'Tylenol' is an 'Acetaminophen'. In this case, tuples t_1 t_2 do not violate F_1.

id	**Symptom**	**Disease**	**Medicine**
t_1	Sore throat	Common Cold	Tylenol
t_2	Sore throat	Flu	Acetaminophen
t_3	Cough	Common Cold	Acetaminophen
t_4	Fever	Coryza	Acetaminophen
t_5	Headache	Common Cold	Ibuprofen
t_6	Lightheadedness	Migraine	Ibuprofen
t_7	Lightheadedness	Hemicrania	Ibuprofen

Table 1.2: Clinical Dataset

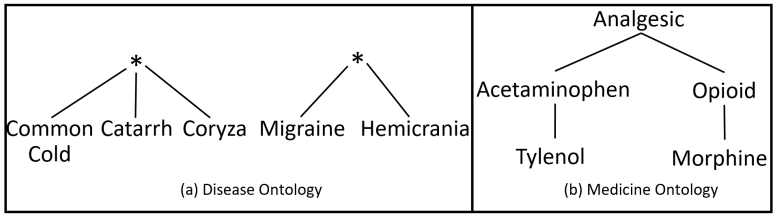


Figure 1.3: Ontologies over a *clinical* dataset.

To capture the data semantics between attribute values, Baskaran et al. present a new class of data quality rules, Ontology Functional Dependencies (OFDs) [9], which extend existing FDs with a set of ontologies, and consider domain relationships between values, namely synonyms. A synonym OFD X Ñ$_{syn}$ A is an extension of an FD *w.r.t.* a synonym ontology, which states that for any two tuples with the same value on attribute X, their values on attribute A should either be the same or are synonyms according to the ontology. For example, for the Disease ontology in Figure 1.3(a), 'Migraine' and 'Hemicrania' are synonyms.

However, data may change over time because of human errors (e.g., typo and incorrect updates). Similarly, the ontology may miss values due to asynchronous updates. For example, the US Food and Drug Administration (FDA) approves new drugs on a monthly cycle, while prescribed drug ontologies do not follow the same update frequency. These changes would lead the data to violate the given OFDs *w.r.t.* the ontologies, and it is often unclear whether the data is incorrect or whether the ontology is incorrect (or both). In practice, we cannot always trust data and ontologies equally. For example, an ontology is reliable since it is often manually specified by a domain expert, but it may be out of date. Also, the reliability of data depends on many factors such as data sources and extraction methods. Thus, practical data cleaning approaches must consider errors in the data as well as errors

in the ontologies, as illustrated by the following example.

Example 3: Consider Table 1.2 with two OFDs: $\phi_1 : Symptom\ \tilde{\mathsf{N}}_{syn}\ Disease$ and $\phi_2 : Medicine\ \tilde{\mathsf{N}}_{syn}\ Disease$ *w.r.t.* a disease ontology (Figure 1.3(a)). Tuples t_1 and t_2 violate ϕ_1 since $t_1\mathsf{r}Disease\mathsf{s}$ $t_2\mathsf{r}Disease\mathsf{s}$, and 'Flu' and 'Common Cold' are not synonyms *w.r.t.* the disease ontology. We could repair the violations by changing the data (i.e., change $t_1\mathsf{r}Disease\mathsf{s}$ to 'Flu' or change $t_2\mathsf{r}Disease\mathsf{s}$ to 'Common Cold'). On the other hand, we could also consider ontology repair by adding 'Flu' to the *Disease* ontology as the sibling of 'Common Cold' to resolve the violation. Similarly, for ϕ_2, tuples t_2 t_4 are violations. We can consider either of the following repairs: (1) conduct data repair by changing $t_2\mathsf{r}Disease\mathsf{s}$ to 'Common Cold' (or 'Catarrh' or 'coryza'); (2) conduct data repair by changing both $t_3\mathsf{r}Disease\mathsf{s}$ and $t_4\mathsf{r}Disease\mathsf{s}$ to 'Flu'; or (3) conduct ontology repair by adding 'Flu' to the *Disease* ontology as the sibling of 'Common Cold'. ∎

We assume that an ontology has been carefully curated by a domain expert, hence, the initial concepts and relationships are trusted. However, as application and domain semantics evolve, ontologies can become stale. Hence, we only consider insertions of new concepts (values) as repairs to an ontology. The above example demonstrates cases where repairing data or an ontology (or both) is a better choice to resolve the violations, and also shows the interaction between repairs. To address this problem, we propose a cleaning algorithm that suggests repairs to the data and to the ontology to resolve these inconsistencies.

1.4 Contributions

This thesis addresses three problems to improve data currency and data consistency:

- Problem 1: Given a relational database and an update history with missing updates, how to identify stale cells when timestamps are missing. Secondly, how can we infer the current values (as repairs of stale values)?
- Problem 2: Potential stale values may exist across a set of replicas in distributed database settings. We study the problem of estimating the currency of returned values from the replicas accurately and efficiently without introducing large overhead during query processing;
- Problem 3: Given a relation, an ontology and a set of data quality rules, identify inconsistencies in the relation and in the ontology w.r.t. the data quality rules. Secondly, compute a minimal number of modifications to resolve these inconsistencies (errors).

This thesis makes the following contributions:

1. Data currency: we present *CurrentClean*, a probabilistic system for detection and cleaning of stale data, which learns both temporal and spatial correlations via past updates and existing data values, to infer data currency. Given an update history, CurrentClean identifies missing updates, quantifies the currency of data values based on their spatio-temporal update patterns, and recommends

two types of temporal repairs to current values. 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. This work has been published in ICDE 2019 [75] and CIKM 2019 [110].

2. Replica currency estimation: we estimate the currency of data retrieved across a set of replicas. We provide theoretical bounds on the confidence of replica currency estimation. Our system computes with a minimum probability p, whether the retrieved replicas are current or stale. Using this confidence-bounded replica currency estimation, we implement a novel DYNAMIC consistency level in the open-source, NoSQL database, Cassandra. Experiments show that the proposed replica currency estimation is effective and efficient. In most tested scenarios, with various query loads and cluster configurations, we show our estimations with confidence levels of at least 0.99. Moreover, the overheads introduced due to estimation scoring and training are low, incurring only 0.76% to 1.17% of the original query processing and replica synchronization time costs, respectively. This work is to be published in ACM SIGMOD 2022 [92].

3. Data consistency: Using a class of data quality rules called Ontology Functional Dependencies (OFDs) [10], we propose techniques to resolve data inconsistencies between the data, OFDs, and an ontology. We define a space of minimal data and ontology repairs based on the number of data changes and the number of changes to the ontologies. We propose OFDClean, a framework for holistically finding a set of minimal repairs such that data satisfies the given OFDs *w.r.t.* the modified ontologies. We demonstrate the effectiveness of our techniques on real datasets, and show that OFDs can significantly reduce the number of

false positive errors in data cleaning techniques that rely on traditional FDs. This work will appear in the ACM Journal of Data and Information Quality 2022 [111]. This work has been published as a student paper in ACM SIGMOD 2021 [108], a workshop paper in ICDE Context in Analytics Workshop 2018 [4] and as a demo paper in EDBT 2018 [109].

1.5 Thesis Organization

This thesis is a sandwich thesis consisting of the conference and journal publications [75, 92, 111]. The remainder of the thesis is organized as follows. In Chapter 2, we present a framework to detect and clean stale data by learning both temporal and spatial correlations via an update history. In Chapter 3, we introduce our models to estimate currency over a set of replica nodes. In Chapter 4, we describe our approach for finding minimal repairs to a relation and ontology with respect to a set of data quality rules. Finally, in Chapter 5, we conclude the thesis with final remarks and directions for future work.

Chapter 2

CurrentClean: Spatio-temporal Cleaning of Stale Data

2.1 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 [11]. *Data currency* is defined as how promptly the data is updated to reflect actual changes in the real world [11]. 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 [17], coupling with master data for improved reliability [44], capturing semantics to maximize accuracy [20, 10], holistic data cleaning systems to improve completeness [83], and human-in-the-loop systems to involve user feedback and the crowd [25]. A surprisingly large percentage of real data is *stale*. A preliminary evaluation using two real datasets containing sensor readings in an IoT smart city [84], and retail transactions [22] 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.

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

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 2.1: Monitoring metrics.

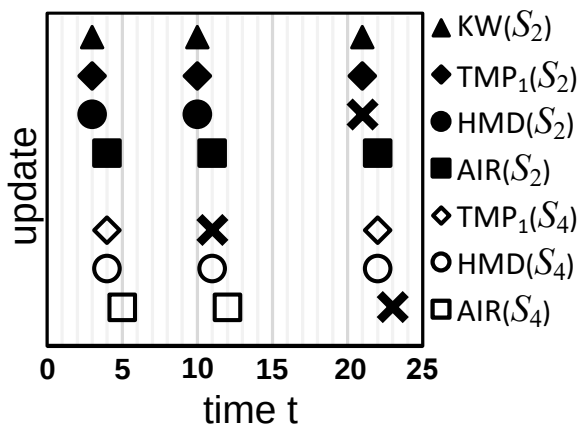


Figure 2.1: Update patterns

missing updates according to the rate of update, and the co-occurrence of past updates, according to an individual behavior pattern.

Example 4: Table 2.1 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 2.1 shows the most recent values of these sensor measurements. ∎

Spatial Correlations. In the presence of timestamps, existing work claims that currency is trivially determined [38]. 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 2.1. Figure 2.1 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 2.1 is available via database transaction logs, or flashback recovery features that allow users to view past states of database objects. Table 2.1 highlights stale values with a black background.

Temporal Correlations. Our discussion thus far considers correlated updates within a time unit. Consider 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 2.1 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 2.1 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 2.1 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 [91, 105], and cleaning imprecise timestamps [106]. 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 [74, 99]. HoloClean uses probabilistic inference to combine signals from multiple sources (external knowledge bases, statistical methods, data dependencies) to determine the most likely repair [83]. These techniques focus on non-temporal models of inference over a static data instance, where cleaning is irrespective of currency [74, 83, 99].

Recent work proposes currency constraints that express currency relationships based on the data semantics [39]. 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 chapter, 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 [36], and proposals for change exploration systems [16], and is complementary to systems like HoloClean [83], 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 2.2-2.3)
- We propose inference rules that model causal and co-occurrence update patterns to estimate currency, and to recommend spatio-temporal aware repairs. (Section 2.4)
- 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 2.5)
- We present three optimizations that prune weakly correlated updates, and improve the inference runtime. (Section 2.6)
- 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 2.7)

2.2 The CurrentClean Framework

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

2.2.1 Problem Definition

A relation R with attributes $A_1, ..., A_n$ is a finite set of n-ary tuples $\mathsf{t}r_1, ..., r_N\mathsf{u}$. A database D is a finite set of relations $R_1, ..., R_m$. We define a *cell* $r\mathsf{r}A_i\mathsf{s}$ 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 $\mathsf{x}q_1, q_2..., q_{|H|}\mathsf{y}$, each q_i $\mathsf{p}t, r\mathsf{r}A\mathsf{s}, \mathsf{a}, \mathsf{bq}\,\mathsf{P}$ H is a query that updates the value of $r\mathsf{r}A\mathsf{s}$ 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. Consequently, the values in the relation contain potentially stale, but values that were correct at some time. We do not consider noise in neither the history nor the relation. Let H be the ground truth sequence of updates, then H „ H. Let q_c and q_c be the last update

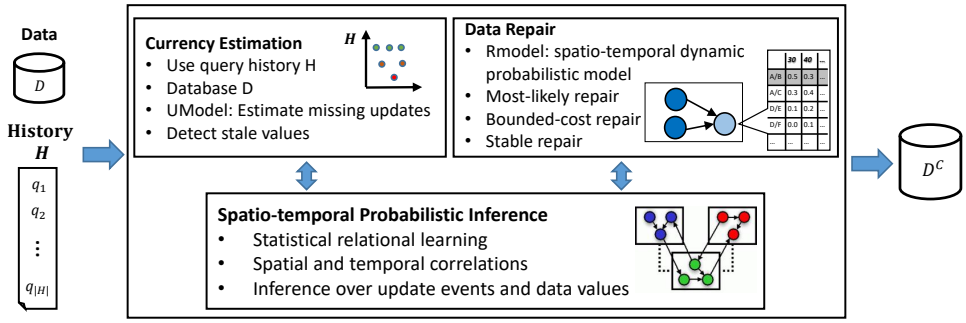


Figure 2.2: CurrentClean system overview.

query over cell c in H and H, respectively. We consider c to be *current* if $q_c.t \quad q_c.t$, otherwise it is *stale*. Intuitively, c is stale if the update $q_c \ \mathsf{R}\ H$.

Error Detection Problem. Since 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., $P\mathsf{p}q_c.t \quad T_c|D, H\mathsf{q}$. Given D and H, the error detection problem is to find a set $\mathcal{E}$ of erroneous (stale) cells in D with $p \quad \beta$, for a currency threshold β.

Data Repair Problem. Let v_c be the unknown true current value of c, and v_c^1 be the estimation of v_c. The data repair problem is to compute the value of v_c^1 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^1 as a repair. We consider a data repair to be correct if $v_c^1 \quad v_c$.

2.2.2 Solution Overview

Figure 2.2 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 2.3.2) 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*) [35]. 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 2.3.3), and define a set of inference rules that model spatio-temporal update patterns (Section 2.4). In our current implementation, we use DeepDive [90] 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 P dompc_{err}q that maximizes Pp$V_{c_{err}}^t$ u|H, Dq. A *bounded repair* restricts the magnitude of change to be within a bound η.

2.3 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.

2.3.1 Dynamic Probabilistic Relational Models

A Probabilistic Relational Model (PRM) is a graphical model for inference over relational data [66]. PRMs model dependencies between attributes under a given schema,

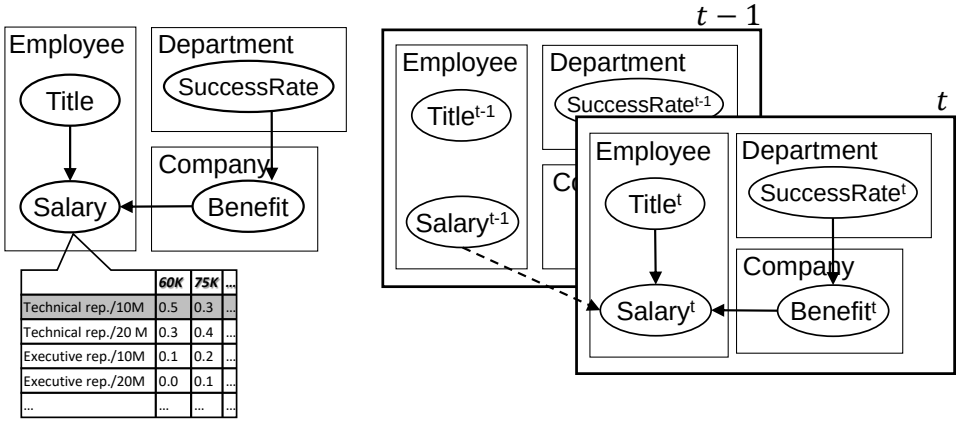


Figure 2.3: (a) PRM CPD (b) 2PRM.

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 2.3(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 [66]. 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 [94]. A DPRM is defined using two PRMs: (i) $\mathcal{M}_0$ is an initial PRM defining the distribution P over D^0, and (ii) $\mathcal{M}^{\tilde{N}}$ is a *two-time-slice PRM* (2TPRM) that connects successive instances D^t to $D^{t\ \ 1}$. Figure 2.3(b) shows a DPRM defined via a 2TPRM representing the transition probability $P\mathfrak{p}D^t|D^{t\ \ 1}\mathfrak{q}$, where an employee's salary at time t is dependent on her title, benefits, and salary at $t\quad 1$.

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 [66]. The complexity of inference lies in the number of random variables and their values. The inference problem is proved to be intractable and hard to approximate [66]. Sampling techniques, such as Gibbs sampling and particle filtering, are used for approximate inference in DPRMs and PRMs [66]. 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 [66].

2.3.2 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 2.4.2, we define a set of update patterns that capture causal and co-occurrence update relationships.

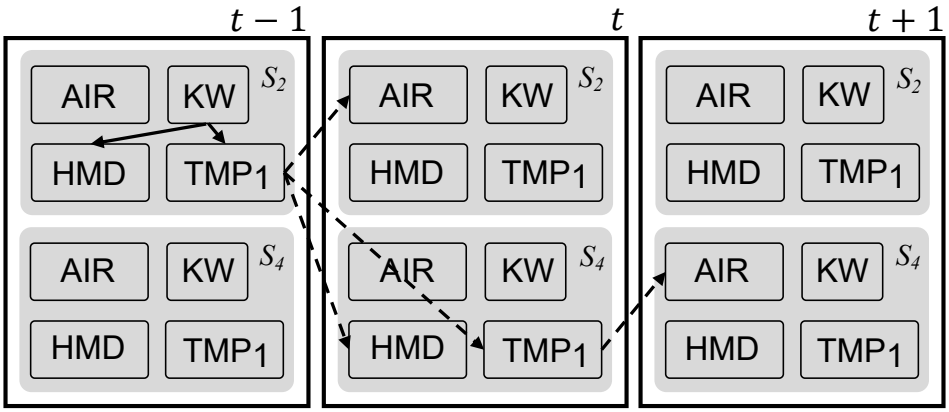


Figure 2.4: Spatio-temporal update chain.

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*, which is a sequence of foreign-key references among the participating tuples.

Definition 3.1: An update event instance I_D^t models updates for each cell c P D at time t. For each c, there is a cell e^t P I_D^t with value trueif c was updated at time t; falseif there is no update; or Nullif 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 $\mathcal{M}^{\tilde{N}}$ defining the transition probability PpI_D^t|I_D^t ^{1}q. I

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 2.4 shows an update chain over three time units for servers S_2 and S_4 from Table 2.1. 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).

2.3.3 Estimating Data Currency

We define a STPM model called the UModel, which 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 (denoted by a random variable $\mathcal{X}$) follows a *Poisson Process* with mean λ [94], where λ is equal to the expected value of $\mathcal{X}$, i.e., λ $E\mathsf{p}\mathcal{X}\mathsf{q}$. 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 to represent the expected time between updates. 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 [107]).

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 2.4.2). 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 $I_D^t, t \mathsf{P} \mathsf{r}0, T\mathsf{s}$ are training data to learn the initial probability distribution $\mathcal{M}_0$, and the transition distribution $\mathcal{M}^{\tilde{\mathsf{N}}}$. We select any instance I_D^t as a baseline for training $\mathcal{M}_0$, whereas, training $\mathcal{M}^{\tilde{\mathsf{N}}}$ 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 $current\mathsf{p}c\mathsf{q}$ represent the probability that c is up-to-date. We estimate $current\mathsf{p}c\mathsf{q}$ 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 trueif c was updated at $t_c \quad i$ (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 $current\mathsf{p}c\mathsf{q} \quad \beta$.

$$current\mathsf{p}c\mathsf{q} \quad P\mathsf{p}E_c^{t_c\ 1} \quad \mathsf{false}, ..., E_c^T \quad \mathsf{false} \mid Pa\mathsf{p}E_c^{t_c\ 1}\mathsf{q}, ..., Pa\mathsf{p}E_c^T\mathsf{qq}. \tag{2.3.1}$$

2.4 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.

2.4.1 DeepDive and DDlog Rules

CurrentClean uses DeepDive, a statistical learning and probabilistic inference engine [90] 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 ψ P $\mathbf{F}$ (ψ „ $\mathbf{X}$) is associated with a function f_ψ and a weight w_ψ. The function f_ψ assigns a value in t 1, 1u 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 Pp$\mathbf{X}$q.

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: $\mathsf{Coworker}(\mathsf{e}_1, \mathsf{e}_2, \mathsf{d})$:- $\mathsf{Employee}(\mathsf{e}_1, \mathsf{d}), \mathsf{Employee}(\mathsf{e}_2, \mathsf{d})$.

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, (2.4.1) defines a variable predicate Raise? modeling the likelihood an employee receives a raise if her coworker received a raise.

$$\mathsf{Raise}(\mathsf{e}_2) \text{ :- } \mathsf{Coworker}(\mathsf{e}_1, \mathsf{e}_2, \mathsf{d}), \mathsf{Raise}(\mathsf{e}_1) \quad [@\mathsf{weight}(\mathsf{w})]. \tag{2.4.1}$$

The head atoms in inference rules can be replaced with simple logical formulas. Rule (2.4.1) can be defined in DDlog by Rule (2.4.2) with an implication in its head:

$$\mathsf{Raise}(\mathsf{e}_1) \Rightarrow \mathsf{Raise}(\mathsf{e}_2) \text{ :- } \mathsf{Coworker}(\mathsf{e}_1, \mathsf{e}_2, \mathsf{d}), [@\mathsf{weight}(\mathsf{w})]. \tag{2.4.2}$$

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 (2.4.1)); or (iii) a function f of variables in the inference rule, e.g. $@\mathsf{weight}(f(\mathsf{d}, \mathsf{q}))$ 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 $\mathsf{SalaryUpdate}(\mathsf{e}, \mathsf{s})$ that determines whether employee e receives a raise based on changes to her salary s.

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

2.4.2 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:

- Cellpc, tid, aidq: cell c in tuple tid in attribute aid.
- DomValuepaid, vq: value v is in the domain of aid.
- Valuepc, vq: cell c contains value v in database D.
- Updatepc, t, vq: c is updated to v at time t w.r.t. H.
- FKeypfkid, kidq: foreign key fkid refers to key attribute kid.
- ExpUpdate?pc, tq: a variable predicate representing boolean random variables E_c^t in UModel modeling whether c is updated at time t.
- Currency?pcq: a variable predicate representing the currency estimation of cell c according to (2.3.1).
- LastUpdatepc, tq: an intensional predicate stating cell c was last updated at time t.
- Linked$\mathsf{pc}_1, \mathsf{c}_2, \mathsf{kq}$: an intensional predicate defining the *relational link* between cells c_1, c_2 with k foreign key references. If $\mathsf{c}_1, \mathsf{c}_2$ 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:

$$\mathsf{Linked}(\mathsf{c}_1, \mathsf{c}_2, 0) \text{ :- } \mathsf{Cell}(\mathsf{c}_1, \mathsf{r}, \text{-}), \mathsf{Cell}(\mathsf{c}_2, \mathsf{r}, \text{-}), \tag{2.4.3}$$

$$\mathsf{Linked}(\mathsf{c}_1, \mathsf{c}_2, 1) \text{ :- } \mathsf{Cell}(\mathsf{c}_1, \mathsf{r}_1, \text{-}), \mathsf{Cell}(\mathsf{cf}, \mathsf{r}_1, \mathsf{fk}), \mathsf{Cell}(\mathsf{c}_2, \mathsf{r}_2, \text{-}), \tag{2.4.4}$$

$$\mathsf{Cell}(\mathsf{ck}, \mathsf{r}_2, \mathsf{key}), \mathsf{FKey}(\mathsf{fk}, \mathsf{key}), \mathsf{Value}(\mathsf{ck}, \mathsf{v}), \mathsf{Value}(\mathsf{cf}, \mathsf{v}).$$

Rule (2.4.3) defines two cells within a tuple. Rule (2.4.4) links cells c_1 and c_2 in tuples r_1 and 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 2.7.2, we evaluate this overhead for increasing k). We define LastUpdate of cell c at t, where MAX is a DeepDive function selecting the max value of t: LastUpdatepc, MAXrtsq:-Updatepc, tq.

Inference Rules. Our inference rules model causality and co-occurrence relationships among updated cells. The first Rule (2.4.5), defines a *positive causality* relationship between c_1 and c_2 via the logical implication in the head, i.e., if c_1 is updated, then c_2 must be updated, in either the same or subsequent time unit. The @weight$\mathsf{p}f_1\mathsf{p}\mathsf{a}_1, \mathsf{a}_2, \mathsf{t}_2\text{-}\mathsf{t}_1\mathsf{qq}$ and the function f_1 are defined according to the instantiation of the rule and the specific values of a_1, a_2, and t_2-t_1.

$$\mathsf{ExpUpdate}(\mathsf{c}_1, \mathsf{t}_1) \text{ => } \mathsf{ExpUpdate}(\mathsf{c}_2, \mathsf{t}_2) \text{ :- } \mathsf{Cell}(\mathsf{c}_1, \text{-}, \mathsf{a}_1), \tag{2.4.5}$$
$$\mathsf{Cell}(\mathsf{c}_2, \text{-}, \mathsf{a}_2), \mathsf{Linked}(\mathsf{c}_1, \mathsf{c}_2, \text{-}), \mathsf{Time}(\mathsf{t}_1), \mathsf{Time}(\mathsf{t}_2), \mathsf{t}_1 \leqslant \mathsf{t}_2 \leqslant \mathsf{t}_1{+}1,$$
$$[\mathsf{@weight}(f_1(\mathsf{a}_1, \mathsf{a}_2, \mathsf{t}_2\text{-}\mathsf{t}_1))].$$

The next Rule (2.4.6) defines a *co-occurrence* update pattern where updates to c_1 must co-occur with updates to c_2. Intuitively, 'c_1 *is updated if and only if* c_2 *is updated.'*

$$\mathsf{ExpUpdate}(\mathsf{c}_1, \mathsf{t}_1) \text{ <=> } \mathsf{ExpUpdate}(\mathsf{c}_2, \mathsf{t}_2) \text{ :- } \mathsf{Cell}(\mathsf{c}_1, \text{-}, \mathsf{a}_1), \tag{2.4.6}$$
$$\mathsf{Cell}(\mathsf{c}_2, \text{-}, \mathsf{a}_2), \mathsf{Linked}(\mathsf{c}_1, \mathsf{c}_2, \text{-}), \mathsf{Time}(\mathsf{t}_1), \mathsf{Time}(\mathsf{t}_2), \mathsf{t}_1 \leqslant \mathsf{t}_2 \leqslant \mathsf{t}_1{+}1,$$
$$[\mathsf{@weight}(f_2(\mathsf{a}_1, \mathsf{a}_2, \mathsf{t}_2\text{-}\mathsf{t}_1))].$$

Rule (2.4.7) models *negative causality* when updates should not occur. Intuitively, in Rule (2.4.7) when c_1 is updated, then 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.

$$\mathsf{ExpUpdate}(\mathsf{c}_1, \mathsf{t}_1) \text{ => !}\mathsf{ExpUpdate}(\mathsf{c}_2, \mathsf{t}_2) \text{ :- } \mathsf{Cell}(\mathsf{c}_1, \text{-}, \mathsf{a}_1), \tag{2.4.7}$$
$$\mathsf{Cell}(\mathsf{c}_2, \text{-}, \mathsf{a}_2), \mathsf{Linked}(\mathsf{c}_1, \mathsf{c}_2, \text{-}), \mathsf{Time}(\mathsf{t}_1), \mathsf{Time}(\mathsf{t}_2), \mathsf{t}_1 \leqslant \mathsf{t}_2 \leqslant \mathsf{t}_1{+}1,$$
$$[\mathsf{@weight}(f_3(\mathsf{a}_1, \mathsf{a}_2, \mathsf{t}_2\text{-}\mathsf{t}_1))].$$

Our model is extensible to capture more complex patterns involving multiple variables. For example, Rule (2.4.8) involves three variable predicates in the head associating updates over cells c_1 and c_2 at time t_1 to updates over c_3 at t_2. The rule body is similar to Rule (2.4.7) except over three cells.

$$\begin{aligned}&\mathsf{ExpUpdate}(\mathsf{c}_1,\mathsf{t}_1) \wedge \mathsf{ExpUpdate}(\mathsf{c}_2,\mathsf{t}_1) \Rightarrow \mathsf{ExpUpdate}(\mathsf{c}_3,\mathsf{t}_2) \text{ :-} \\ &\mathsf{Cell}(\mathsf{c}_1, \text{-}, \mathsf{a}_1), \mathsf{Cell}(\mathsf{c}_2, \text{-}, \mathsf{a}_2), \mathsf{Cell}(\mathsf{c}_3, \text{-}, \mathsf{a}_3), \mathsf{Linked}(\mathsf{c}_1, \mathsf{c}_3, \text{-}), \\ &\mathsf{Linked}(\mathsf{c}_2, \mathsf{c}_3, \text{-}), \mathsf{Time}(\mathsf{t}_1), \mathsf{Time}(\mathsf{t}_2), \mathsf{t}_1 \leqslant \mathsf{t}_2 \leqslant \mathsf{t}_1{+}1, \\ &\qquad\qquad [\mathsf{@weight}(f_4(\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \mathsf{t}_2\text{-}\mathsf{t}_1))].\end{aligned} \tag{2.4.8}$$

We note that contradictions can occur among these rules. For example, negative causality rules such as (2.4.7) may contradict positive causality rules of the form (2.4.5). 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 ExpUpdate: ExpUpdatepc, tq = true :- Update$\mathsf{pc}, \mathsf{t}, \text{-}\mathsf{q}$. According to the rule, if there is an Update for c at time t, then ExpUpdatepc, tq is true, otherwise it is set to false.

To compute *current*$\mathsf{pc}\mathsf{q}$, Rule (2.4.9) models the likelihood there are no updates since the last time c was updated (at t_1) until the current time. We use t_2 to range over this time period. In DeepDive, we use the Time and ALL predicates to capture

all times t_2 where there is no expected update since t_1.

$$\begin{aligned}&\mathsf{Current}(\mathsf{c}) \text{ :- } \mathsf{LastUpdate}(\mathsf{c}, \mathsf{t}_1),\\ &\qquad \mathsf{ALL}[!\mathsf{ExpUpdat}(\mathsf{c}, \mathsf{t}_2), \mathsf{t}_1 < \mathsf{t}_2, \mathsf{Time}(\mathsf{t}_2)].\end{aligned} \tag{2.4.9}$$

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 [90]. 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.

2.5 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.

2.5.1 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 P r0, Ts. We populate each instance D^t based on the queries in H, i.e., for each cell c^t P 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 [17].

Most-Likely Repairs (MLR). The *most-likely repair* for a stale cell c_{err} P $\mathcal{E}$ 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 Pap$V_{c_{err}}^T$q. For c_{err} in attribute A, the repair value v is chosen from the domain of A as follows.

$$\arg \max_{v \mathsf{P} Dom \mathsf{p} A \mathsf{q}} \mathsf{p} P \mathsf{p} V_{c_{err}}^T \quad v | Pa \mathsf{p} V_{c_{err}}^T \mathsf{qqq}.$$

For example, the most-likely repair for S_2 HMD in Table 2.1 is dependent on its TMP_1 and KW values.

Bounded-Cost Repairs (BCR). Complementary to existing minimal-change and cardinality-minimal techniques [17], we propose *bounded-cost repairs* that restrict the magnitude of change to c_{err} according to a distance metric d such that $d\mathsf{p}v, v_{c_{err}}\mathsf{q}$ ¤ η where $v_{c_{err}}$ is the stale value in c:

$$\arg \max_{v \mathsf{P} Dom_{\eta, v_{c_{err}}} \mathsf{p}A\mathsf{q}} \mathsf{p}P\mathsf{p}V^T_{c_{err}} \quad v|Pa\mathsf{p}V^T_{c_{err}}\mathsf{qq}.$$

$Dom_{\eta, v_{c_{err}}}\mathsf{p}A\mathsf{q}$ is a subset of values in $Dom\mathsf{p}A\mathsf{q}$ within a distance of η from $v_{c_{err}}$. For example, if η 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.

2.5.2 Translating Values to Inference Rules

To instantiate the RModel, we define DDlog inference rules to infer repairs. We use the predicates defined in Section 2.4.2 and define a variable predicate ExpValue?pc, t, vq that specifies the query random variables V^t_c.

$$\begin{aligned}&\mathsf{ExpValue}(\mathsf{c_1}, \mathsf{t_1}, \mathsf{v_1}) \Leftrightarrow \mathsf{ExpValue}(\mathsf{c_2}, \mathsf{t_2}, \mathsf{v_2}) \text{ :- } \mathsf{Cell}(\mathsf{c_1}, \text{-}, \mathsf{a_1}),\\ &\mathsf{Cell}(\mathsf{c_2}, \text{-}, \mathsf{a_2}), \mathsf{Linked}(\mathsf{c_1}, \mathsf{c_2}, \text{-}), \mathsf{Time}(\mathsf{t_1}), \mathsf{Time}(\mathsf{t_2}), \mathsf{t_1} \leqslant \mathsf{t_2} \leqslant \mathsf{t_1}+1,\\ &\mathsf{DomValue}(\mathsf{a_1}, \mathsf{v_1}), \mathsf{DomValue}(\mathsf{a_2}, \mathsf{v_2}), \ [\texttt{@weight}(f_5(\mathsf{a_1}, \mathsf{a_2}, \mathsf{t_2}\text{-}\mathsf{t_1}))].\end{aligned} \tag{2.5.1}$$

Rule (2.5.1) 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: ExpValuepc, t, vq = true :- Updatepc, t, vq.

Algorithm 1: *CurrentClean*pD, Hq

Input : Database D and history H

1 UModel Ð *buildCurrencyModel*pD, Hq;
2 $\mathcal{E}$ Ð H; D^C Ð D;
3 **for** c P D^C **do**
4 | **if** UModel.*current*pcq β **then** $\mathcal{E}$ Ð $\mathcal{E}$ Y tcu;
5 RModel Ð *buildRepairModel*pD, Hq;
6 **for** c_{err} P $\mathcal{E}$ **do** c_{err}.value Ð RModel.*repair*pc_{err}q ;
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 ExpValuepc, t, vq. We give details of CurrentClean's repair algorithm in Algorithm 1.

2.5.3 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 $|D|$ $|H|$ [66]. 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 logpn_vq: (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 (trueand 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.

2.6 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 (2.4.5), we replace $\mathsf{ExpUpdate}_{\mathsf{pc}_1}, \mathsf{t}_1\mathsf{q}$ with $\mathsf{Update}_{\mathsf{pc}_1}, \mathsf{t}_1\mathsf{q}$, and move the predicate to the body. This allows our model to compute $\mathsf{ExpUpdate}_{\mathsf{pc}_2}, \mathsf{t}_2\mathsf{q}$

directly from the history. The resulting rules contain one variable predicate that allows DeepDive to perform efficient Gibbs sampling [90], and reduces the inference complexity from NP-hard to $n_v \quad \log\mathsf{p}n_v\mathsf{q}$. Our evaluation shows 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 $A_i, A_j \mathsf{P} \mathbf{R}$. 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 $p \quad 95\%$ confidence, and approximately one-third of the attribute pairs are pruned. These attribute pairs represent low-weight update patterns in DeepDive. In Section 2.7.3, 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 $\mathsf{Irrelevant}\mathsf{p}A_i, A_j\mathsf{q}$ 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., [62].

This dynamic programming partitioning (DPP) algorithm computes optimal bucket boundaries for a set of error metrics in quadratic time w.r.t. 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*, $R^2 \quad 1 \quad \frac{SSE}{SST}$, 0 ¤ R^2 ¤ 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 80-95%.

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 [2]. We define a d-cube over three dimensions: attributes A_i and A_j, and time intervals t. Figure 2.5 shows a d-cube where a cell is

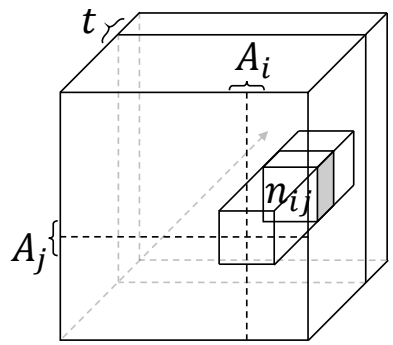


Figure 2.5: d-cube.

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 according to 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 2.5 containing cell n_{ij}. For a time t, to quantify the association $\mathfrak{p}A_i, A_j\mathfrak{q}$ based on their update frequency, we use the *Normalized Pointwise Mutual Information* (NPMI) measure defined as [62]:

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

where $P\mathfrak{p}A_i, A_j\mathfrak{q}$ is the joint update probability of $\mathfrak{p}A_i, A_j\mathfrak{q}$, and $P\mathfrak{p}A_j\mathfrak{q}$ 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 $npmi\mathfrak{p}A_i, A_j\mathfrak{q}$ 1 indicates no co-updates, 0 indicates independence, and 1 indicates $\mathfrak{p}A_i, A_j\mathfrak{q}$ are always updated together. For a given threshold α,

we prune updates $\mathsf{p}A_i, A_j\mathsf{q}$ where $npmi\mathsf{p}A_i, A_j\mathsf{q} \quad \alpha$. Our evaluation shows that for $\alpha \quad 0.3$, Opt-3 improves runtime by 70% with a 1.8% accuracy loss.

2.7 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.

2.7.1 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. For the experiments evaluate the effectiveness of optimizations (Section 2.7.3), the reported runtimes are the average of three executions due to the excessively long time it takes to learn update patterns over the whole update history with large domain size for each iteration. For all other experiments, reported runtimes are the average of six executions.

Datasets. We use four real data collections. Table 2.2 gives the data characteristics,

Table 2.2: Real data characteristics.

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

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 [63]: 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 [26]: 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) [22]: 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 [54]: 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 ¤ TMP ¤ 37.5 C, and SpO2 ¥ 94%. The full set of attribute update dependencies and value ranges can be found in [107]. 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) [39]: 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 $r_1[\mathsf{Status}]$ = *retired* is more recent than $r_2[\mathsf{Status}]$ = *employed* for the same person. We define a set of currency constraints based on temporal relations from domain experts (we refer the reader to [107] for details). For example, for tuples $r_1[\mathsf{BTAN}]$ and $r_2[\mathsf{BTAN}]$ describing #blood tests for a patient in the Mimic data, if $r_1[\mathsf{BTAN}]$ ¡ $r_2[\mathsf{BTAN}]$, 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 [83]: 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 [90]. We input to HoloClean: (i) a set of denial constraints translated from the given update dependencies [107]; (ii) external reference sources such as the National Drug Code Directory [95]; and (iii) we profile the data to obtain statistical frequency distributions of each attribute's domain.

ERACER [74]: 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 Null) by learning the CPDs. We provide ERACER with 11 attribute dependency templates (listed in [107]) 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.

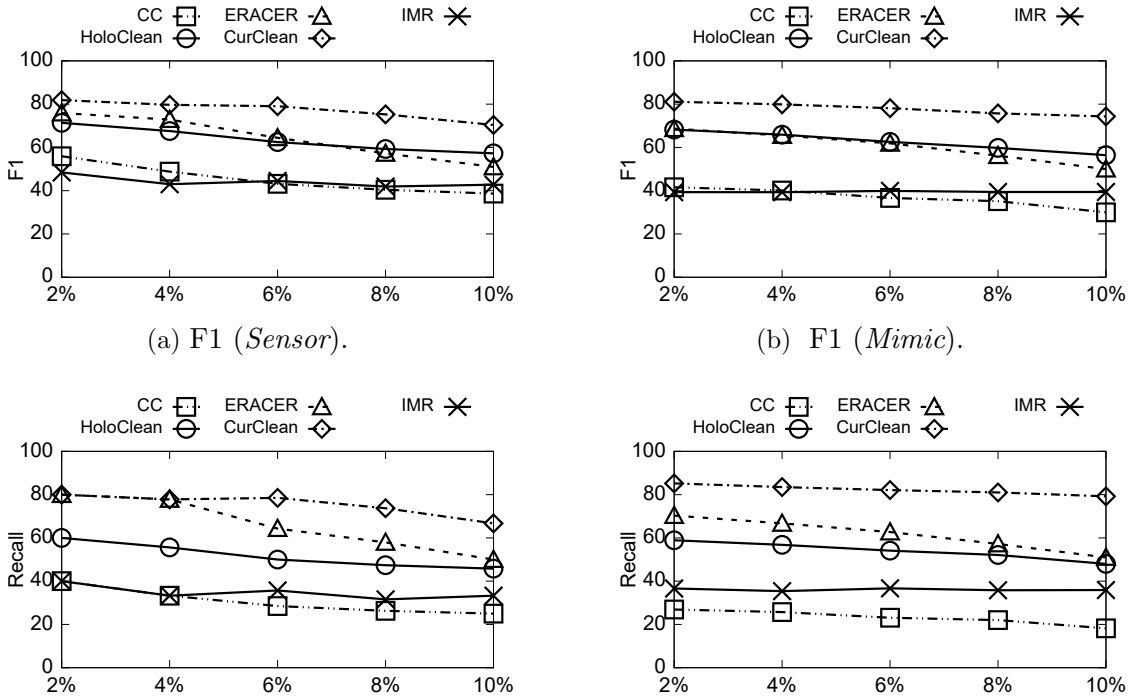
IMR [105]: Given a sample of labeled values, the *Iterative Minimum Repair* (IMR) framework repairs time-series data by recommending updates that differ minimally

Table 2.3: 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
$\|H\|$	history (%)	**32** (Mimic); **40** (Sensor) **35** (Trans); **24** (NBA)

from the labeled truth. High confidence repairs from past iterations are used to determine subsequent repairs [105]. 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 γ p1 eq for a given error rate e, and label cells using the ground truth domain.

Parameters. Unless otherwise stated, Table 4.4 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 δ λ, where λ is the Poisson distribution mean as described in Section 2.3.3. 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.



(a) F1 (*Sensor*).

(b) F1 (*Mimic*).

(c) Recall (*Sensor*).

(d) Recall (*Mimic*).

Figure 2.6: Error detection accuracy for varying error rate e.

2.7.2 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 2.6 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 2.6b and 2.6d 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 2.6(c) and (d) show that CurrentClean outperforms all four methods in recall by an average +34% as many stale values are not captured. ERACER exhibits the greatest sensitivity to increasing e, which negatively influences 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 2.7 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 2.8 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 ¥ 2, only result in a modest +2% gain. Figure 2.9 shows the error detection runtime where k ¤ 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 2.10. 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 ¤ 2, whereas longer update chains occur (Mimic, NBA) when updates are not as localized, containing 5+ values.

2.7.3 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.

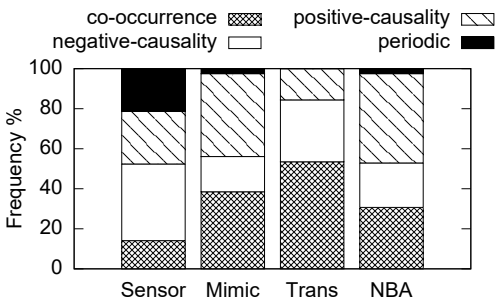


Figure 2.7: Pattern freq.

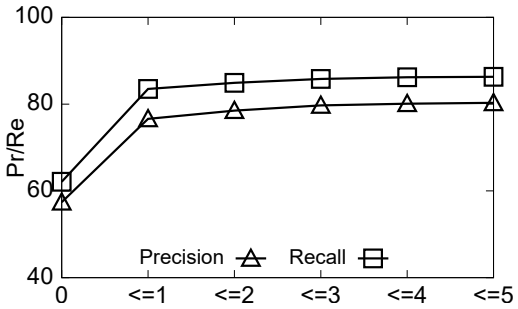


Figure 2.8: Accuracy vs. k

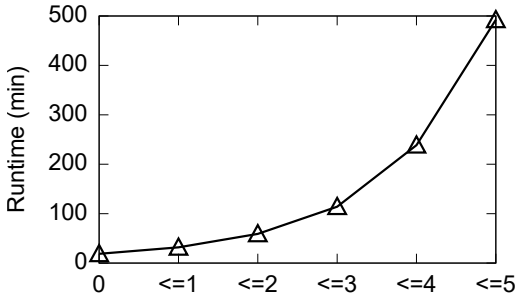


Figure 2.9: Runtime vs. k

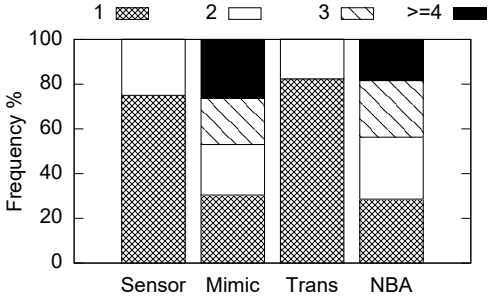
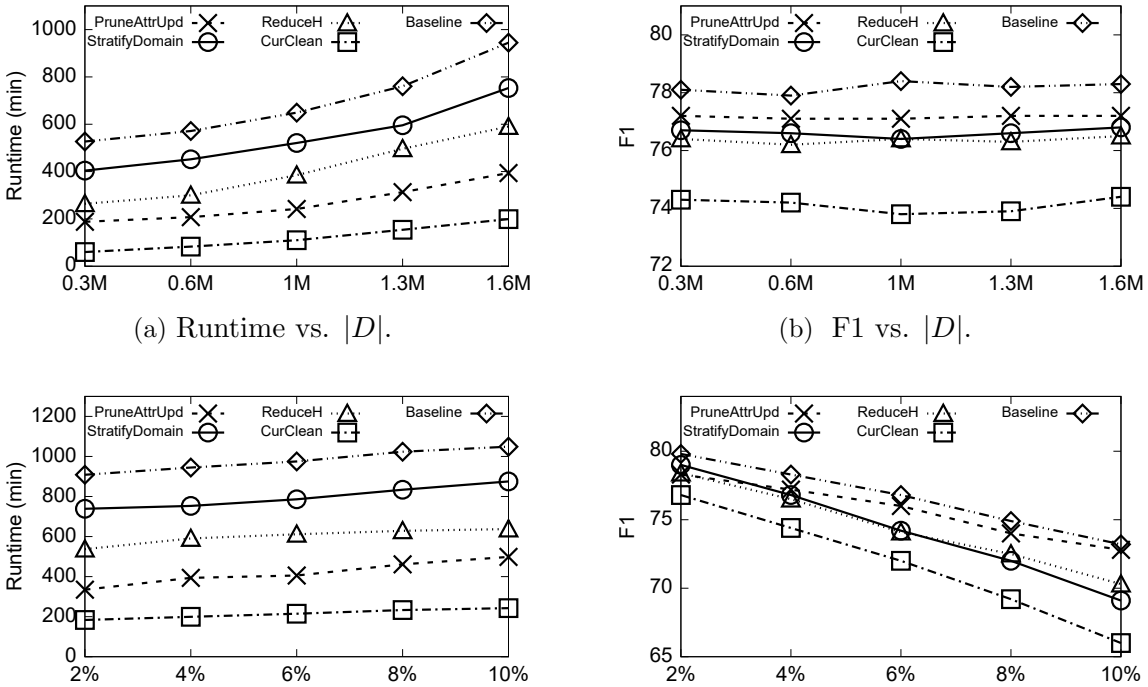


Figure 2.10: Upd. chain freq.

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 2.11a-2.11d 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.



(a) Runtime vs. $|D|$.

(b) F1 vs. $|D|$.

(c) Runtime vs. e.

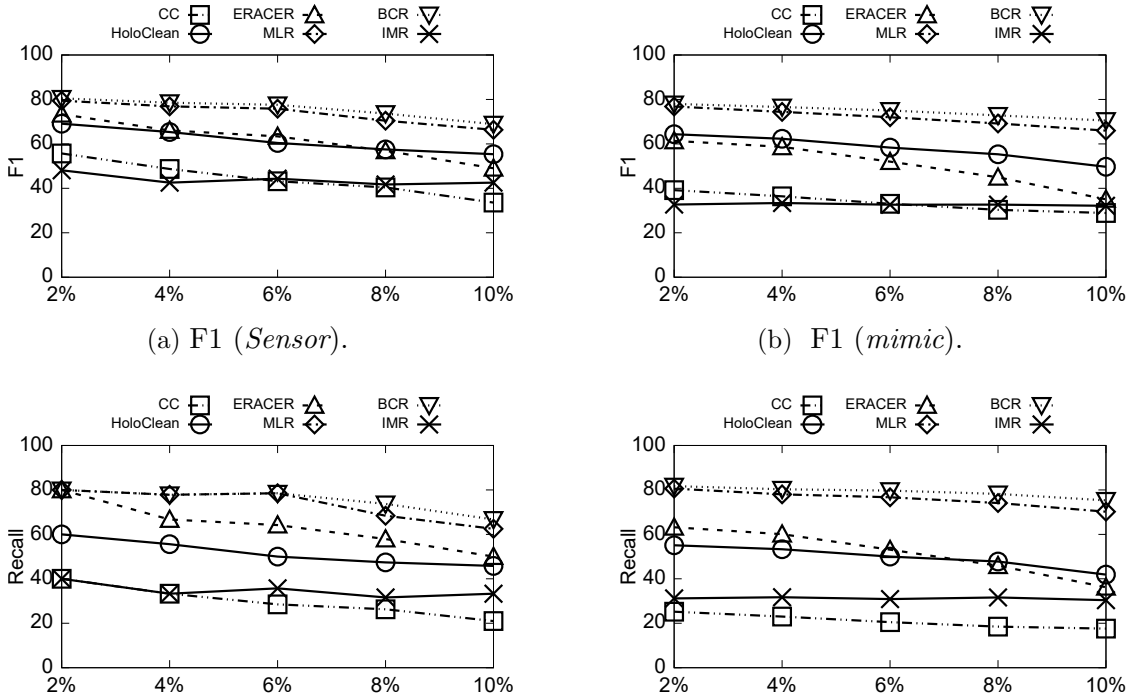
(d) F1 vs. e.

Figure 2.11: Error detection accuracy for varying error rate e.

2.7.4 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 2.12 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 2.12 shows that BCR and MLR achieve superior F1 and recall scores (by +29% and +26%, respectively, over existing techniques), confirming that learning



(a) F1 (*Sensor*).

(b) F1 (*mimic*).

(c) Recall (*Sensor*).

(d) Recall (*Mimic*).

Figure 2.12: Error detection accuracy for varying error rate e.

from past value correlations is needed. CC performs poorly, as relying on hard constraints captures on average 30% of the correct repairs (Figure 2.12c and Figure 2.12d). 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 correlations among the attribute values, resulting in a different and limited set of repair values.

The update history H may exhibit varying distributions, including short bursts of updates followed by lull periods. In such cases, this can influence the confidence of the learned update patterns by the UModel, and the consequent repairs in the

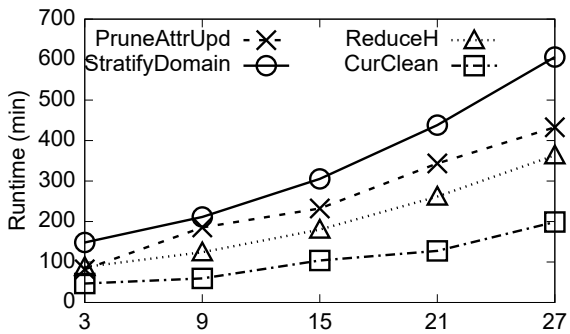


Figure 2.13: Scalability w.r.t. n.

RModel.

2.7.5 Runtime Performance

Exp-7: Scalability. Figure 2.13 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 2.11a 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 2.11c), and $|H|$ (due to Opt-3, we refer to [107] for resulting graphs). These results align with CurrentClean's (optimized) complexity of $n_v \quad \log\mathfrak{p}n_v\mathfrak{q}$. Lastly, as expected, Figure 2.9 shows the exponential scale-up w.r.t. k, due to the complex relationships among the random variable values.

Exp-8: Comparative Performance. Table 4.5 shows the total wall-clock comparative runtimes. CurrentClean incurs additional time due to the combinatorial

Table 2.4: Comparative runtimes.

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

Table 2.5: Example update patterns.

Update pattern	**F1**
(i) Δ WBC Ñ ΔTMP (+ve)	+2.5
(ii) Δ RBC Ñ Δ SpO2 (+ve)	+2.3
(iii) Δ MONO Ø Δ WBC (co-occur)	+2.8
(iv) Δ Hb Ñ (Δ TMP) (-ve)	+0.4

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.

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 Ñ ΔRBC contributed 5% and 7% towards the UModel

and RModel F1, respectively (hemoglobin carries oxygen in red blood cells).[1]

Table 2.5 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.

2.8 Related Work

Temporal-based Cleaning. Currency constraints express currency relationships via pre-defined constraints based on the data semantics [39]. Our evaluation has shown that relying on fixed constraints does not capture temporal and spatial dependencies that exist among updates and their values. Recent work has explored cleaning time-series data that define *speed constraints* to identify and repair values based on expected rates of change [91], applying minimum changes given labeled

[1] The rule $\Delta\mathsf{Hb} \rightarrow \Delta\mathsf{RBC}$ is an instance of (2.4.5), 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.

ground truth [105], repairing imprecise timestamps [106], and generating repairs according to maximum likelihood estimation [104]. All these techniques clean univariate 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 [2]. Temporal dependence is used to understand the behaviour of a user cohort to predict their future behaviour [19]. This *recurrent cohort analysis* identifies causative and dependent behaviour. In a 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 2.7.1, ERACER [74] and HoloClean [83] learn from given attribute templates and training samples to infer clean values. Our evaluation has shown that neither approach recognizes update patterns across the database cells, and they both 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 [48]. 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 [99]. 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.

2.9 Conclusion and Future Work

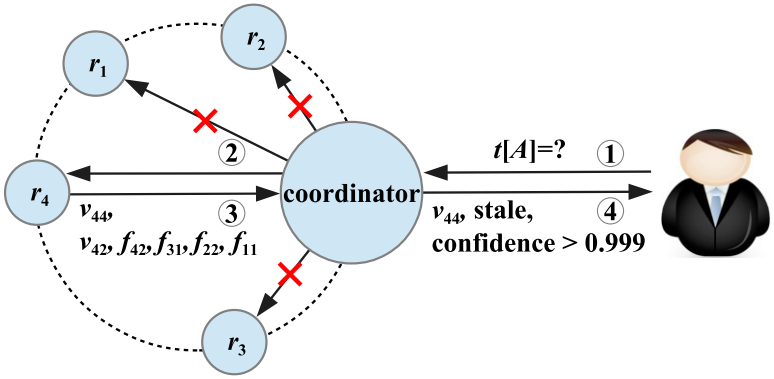
We present CurrentClean, a probabilistic system for detection and repair of stale values. We argue that currency is a relative notion dependent on an individual entity's update patterns. We propose update and repair models, and instantiate them using inference rules that reason over current and past 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.

Chapter 3

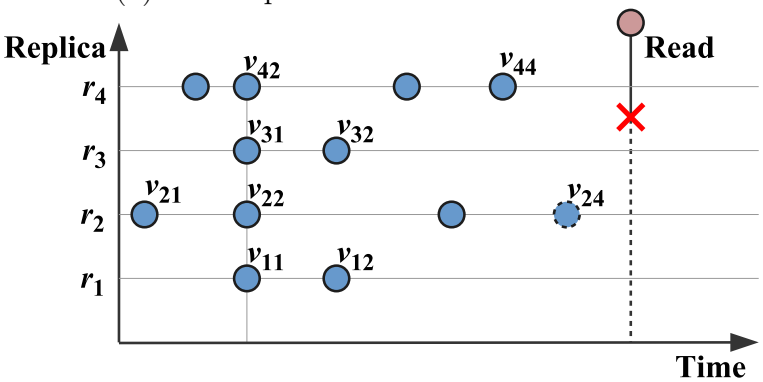
Confidence Bounded Replica Currency Estimation

3.1 Introduction

Data in real systems becomes stale rapidly. Studies have shown that customer records decay at a rate of 70% per year [45]. Different values of the same entity arise if update queries are not applied simultaneously over all replicas. While each of these values was correct at some time, answering queries across these multiple value versions is challenging. Data currency is often synonymously referred to as data freshness in distributed database settings, and a critical data quality dimension [38]. One of the primary problems in data currency is to determine whether a value is stale, and if so, to compute its current value. Different models of currency have been proposed;



(a) Four replicas in a cluster of nodes.



(b) Versions v_{ij} of a cell written to four replicas at different times.

Figure 3.1: A cluster with replication factor 4 runs across multiple data centers. Consistency indicates that a read returns the most recently written version, i.e., v_{24}, the 4th version stored in replica r_2. Unfortunately, this is not always possible due to system availability and network partitioning. It is highly demanded to estimate whether the response replica, e.g., v_{44}, is current or stale.

including declarative currency constraints to determine partial currency order among records [39], and fixed vs. adaptive replication schemes to propagate updates leading to varying freshness guarantees [13, 80, 103].

In this study, we focus on distributed data stores with replication, such as BigTable [21], HBase [50], Dynamo [34], and Cassandra [69], carrying different versions of a data item (i.e., a key-value pair, or a cell in a table), where timestamps are used

to determine their currency. Figure 3.1(a) shows a system with replication factor 4. A cell $t\mathsf{r}A\mathsf{s}$ in row t and column A of a table is replicated across four nodes in the cluster. According to Brewer's CAP theorem [18], guarantees in system availability and partition-tolerance lead to sacrifices in consistency with different consistency levels across the replicas. We study the problem of stale data detection across a set of distributed nodes with varying consistency levels. We adopt a notion of currency, whereby if a pending update exists that has not yet materialized at a node, the replica is considered stale; otherwise, it is fresh/current [68, 75].

By manually specifying a consistency level, existing systems such as Cassandra wait for a fixed number of replicas (ONE, QUORUM, ALL) to respond; such a strategy is ineffective and inefficient. At consistency level ONE, as illustrated in Figure 3.1, the returned replica may be stale. Alternatively, if we adopt the consistency level ALL, waiting for all replicas to respond, it blocks the read request (given the network partitioning in the cluster). The QUORUM level achieves a compromise by retrieving more than half of the replicas. However, the aforesaid issues regarding waiting for a fixed number of replicas, unfortunately, still exist.

Example 5: Consider a vehicle sensor monitoring application at our industrial partner. The workload is write-intensive with a mean inter-arrival of 2 ms, requiring write consistency ONE. The application must retrieve the current engine oil pressure values to trigger safety alarms. In such critical settings, having high confidence, current and accurate readings is essential. Relying on consistency level ONE may trigger false positive alerts, e.g., in Cassandra, and using fixed ALL and QUORUM levels leads to unnecessary delays.

Figure 3.1(a) shows a system with replication factor 4. A cell t[A] in row t, and column A of a table denoting the oil pressure value is replicated in four nodes in the cluster. Let v_{ij} denote the j-th version of a cell in the i-th replica. Figure 3.1(b) shows the latest version v_{24} of the oil pressure value is stored only in replica r_2 with write consistency level ONE, due to write efficiency or network partitioning tolerance. Read requests are also processed at different consistency levels, again, for either read efficiency or network failure. With read consistency level ONE, Cassandra immediately returns the version held by the first node that responds to the query [69], i.e., v_{44} of replica r_4. We can see that version v_{44} is stale, given the more recent v_{24}. However, without accessing the other replicas, r_1 r_3, the system cannot determine whether the retrieved version v_{44} is current or stale, hindering the real-time monitoring of oil pressure values. ∎

In this chapter, we offer a more intelligent option, and propose a new consistency level DYNAMIC, such that the system dynamically determines the number of replicas to retrieve. Specifically, we estimate whether the returned replicas are current or stale, and we provide a guaranteed confidence (with at least probability p) of the estimation.

If the retrieved versions are known to be current with a high probability, we can directly return these values as query results. In contrast, if the values are likely to be stale, then users may decide that it is worthwhile to wait for the remaining replicas. It is worth noting that reading the latest version (using strict consistency) is not always possible due to the underlying network partitioning scheme. With the foundations of machine learning, we study and provide a lower bound p of the confidence estimation to determine whether a returned query answer is current or

stale. This enables systems to adaptively decide whether to incur additional latency costs according to application requirements.

3.1.1 Challenges

We study the problem of computing a lower confidence bound for replica currency estimation, with the following challenges. (1) While efficient learning models are directly applicable, we expect a guaranteed confidence for the currency prediction. It is highly non-trivial to derive a lower bound for the probability of whether the replica is current or stale, with respect to the ground truth. (2) The problem becomes even harder, given that the replicas are distributed across cluster nodes with limited synchronization. Our currency prediction models need to be learned locally in each replica node, such that they are small enough to propagate to the other nodes with low overhead.

3.1.2 Contributions

Our major contributions in this study are as follows.

(1) We derive a theoretical bound on the confidence that a replica is current or stale, with respect to the ground truth (Section 3.2). The derivation is first based on ideal scenarios, where the model is learned over the full update history across all replicas [1].

[1] This contribution is collaborative work with Y. Sun and S. Song, Tsinghua University.

(2) We extend the theoretical bound for the replica currency estimation confidence to realistic scenarios in Section 3.3. We develop models that are learned locally within each replica node, and together with the data, sent to the coordinator node for query processing and confidence-bounded currency estimation [2].

(3) We implement[3] a novel consistency level DYNAMIC, in Section 3.4. The system immediately returns the response replicas if the currency estimation confidence is greater than a threshold η. Instead of the three fixed options (ONE, QUORUM, ALL), the proposed DYNAMIC consistency level offers more finely tunable confidence levels (e.g., η 0.99, 0.999, 0.9999, ...) to evaluate the trade-off between replica currency and query efficiency [4].

(4) We conduct extensive experiments with various query loads and cluster configurations. In most tested scenarios, our replica currency estimation demonstrates high confidence levels (at least 0.99), and is efficient (incurring only 0.76%-1.17% of the original query processing and replica synchronization time costs). Remarkably, the proposed DYNAMIC consistency level achieves higher accuracy to return current answers than QUORUM (comparable to ALL), while keeping latency times low (close to ONE).

Table 3.1 lists frequently used notations.

[2]This contribution is built by Y. Sun and S. Song, Tsinghua University.

[3]Our present implementation uses Cassandra [5], and we believe that the replica currency estimation extends similarly to other (NoSQL) databases, such as HBase [6], which we intend to implement in the future.

[4]This contribution is collaborative work with Y. Sun and S. Song, Tsinghua University.

Table 3.1: Notations.

Symbol	Description
r_i	the i-th replica of the data item (queried by the user)
v_{ij}	the j-th version of the i-th replica for the data item
T_{ij}	time stamp of replica version v_{ij}
t_{ij}	time distance of replica version v_{ij} to the previous version $v_{i,j\ \ 1}$, i.e., $t_{ij} \quad T_{ij} \quad T_{i,j\ \ 1}$
ζ	the current time of user query to the database
f	model for predicting t_{ij}
ϕ	parameter of model f
Z	a number of z samples for training model f

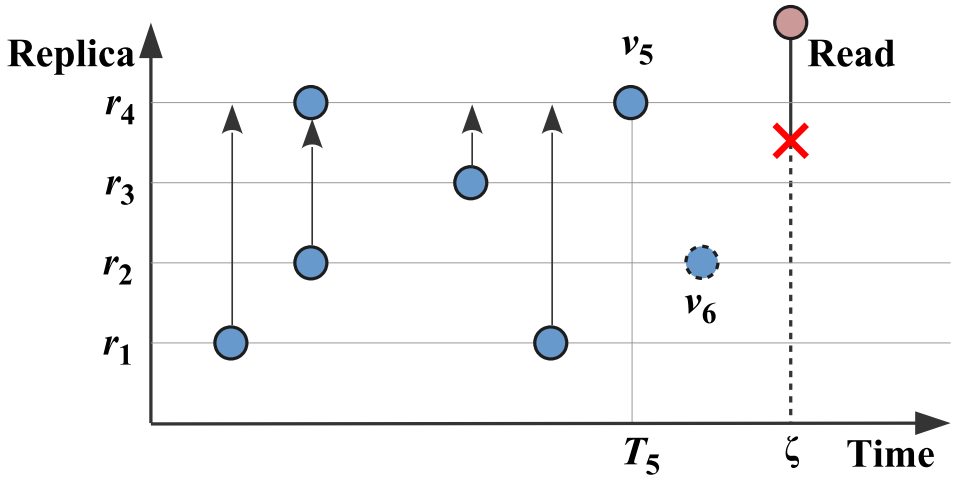
3.2 Ideal Scenarios

We first consider an ideal scenario, where the full update history across all replicas is available, and is propagated to the responding node. Although this is costly in practice, we consider this scenario for the following reasons:

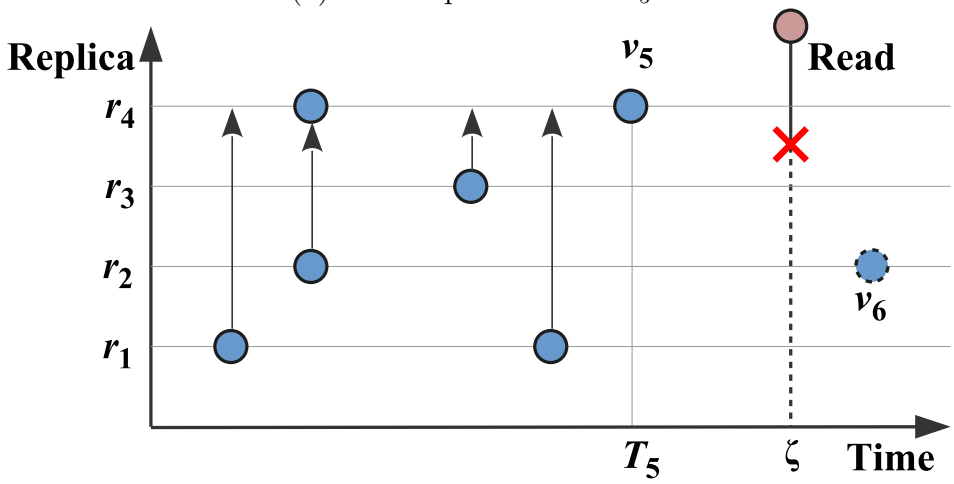
(1) By predicting a bounded confidence over the full history with an ideal node, we can extend this to use a partial history in a realistic node, as presented in Section 3.3. (2) Prediction using an ideal node (by propagating the full write history at no cost) serves as a baseline of the best effort results. This enables us to demonstrate the accuracy of our estimation models (in practice) to the ideal case (Section 3.5).

3.2.1 Replica Currency in Ideal Scenarios

Let v_n with timestamp T_n be the most recent version of the queried data item in the response nodes, and ζ be the query time. Replica currency is determined by the next



(a) Stale replica version v_5



(b) Current replica version v_5

Figure 3.2: Replica currency estimation in an ideal scenario. The update history for all replicas is propagated to a responding node for learning and currency estimation.

version $v_{n\ \ 1}$ with timestamp $T_{n\ \ 1}$, which may not have been retrieved yet from the remaining replicas, or is beyond the query time. If

$$T_{n\ \ 1} \ \text{¤}\ \zeta, \tag{3.2.1}$$

the version v_n is stale, i.e., a more recent version $v_{n\ \ 1}$ is written to the other replicas but not yet retrieved. Otherwise, if

$$\zeta \quad T_{n\ \ 1}, \tag{3.2.2}$$

we can conclude that v_n is current, i.e., the next write operation has not yet occurred and will happen beyond the query time.

Example 6: Figure 3.2(a) shows that version v_5 is the latest version with timestamp T_5 from responding replica r_4. However, if there is a version v_6 written to replica r_2 with timestamp $T_6 \quad \zeta$ that has not responded, then v_5 is a stale version. In contrast, if there is no write between T_5 and ζ, i.e., the next version v_6 does not occur, then the retrieved version v_5 is current. ❙

Example 6 shows two cases should be considered to determine the currency of a data item v_n: (i) whether the next version $v_{n\ \ 1}$ exists in a replica, but has not been retrieved; or (ii) $v_{n\ \ 1}$ has not yet occurred (been written) w.r.t. the current query time, i.e., the corresponding time stamp $T_{n\ \ 1}$ is unknown. Thus, we can evaluate replica currency by estimating the predicted $T^1_{n\ \ 1}$ from $T_{n\ \ 1}$. By bounding the generalization error in the prediction (Proposition 1), we derive a lower bound on the confidence for replica currency estimation of $T^1_{n\ \ 1}$ (Propositions 2 and 3).

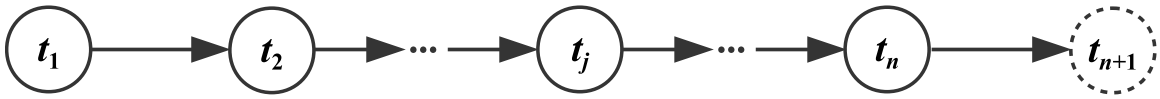


Figure 3.3: Prediction model for consecutive replica versions.

Prediction Model

Let v $\{v_1, \ldots, v_k, \ldots, v_n\}$ be the replica versions that are ideally propagated to the response node of replica r_i. Rather than directly predicting the timestamp of the next version v_{n+1}, as illustrated in [30, 33], it is more reasonable to train a model over the time distance to the last version. That is, we consider the time distance $t_k = T_k - T_{k-1}$ of replica version v_k to the previous version v_{k-1}. Figure 3.3 illustrates the Bayesian model for prediction [49]. The prediction model f predicts t_k referring to the previous time distance t_{k-1},

$$f(t_{k-1}) \approx t_k, \tag{3.2.3}$$

having $t_k = f(t_{k-1}) + \varepsilon$, where ε is the error term. We use a simple linear regression model [98] as f, for the following reasons,

$$t_k^1 = \phi_1 t_{k-1} + \phi_0, \tag{3.2.4}$$

where ϕ_0 and ϕ_1 are the parameters. (1) The generalization error of the linear regression model is bounded [98], as shown in Proposition 1. This serves as a foundation for the confidence-guaranteed replica currency estimation in Section 3.2.2. (2) The model is small enough to be propagated efficiently among the nodes in a cluster. This will

be particularly important in the realistic scenarios considered in Section 3.3, and in our Cassandra implementation in Section 3.4. (3) Linear regression can be efficiently trained, incrementally, as new versions arrive [81].

Consider a sample of training data $Z = \{(t_1, t_2), \ldots, (t_{n-1}, t_n)\}$ obtained from the write history v of replica versions, for learning the model parameters in f. Let $X = \begin{bmatrix} t_1 & t_{n-1} \\ 1 & 1 \end{bmatrix}$ and $Y = \begin{bmatrix} t_2 \\ \vdots \\ t_n \end{bmatrix}$. The linear regression parameters can be learned by setting $\phi = (X^T X)^{-1} X^T Y$. As new versions arrive, v_{n+1}, this leads to an increment $Z^1 = (t_n, t_{n+1})$ of training data, the new parameters w.r.t. $Z \cup Z^1$ are incrementally updated

$$\phi^1 = (X^T X - X^{1T} X^1)^{-1} (X^T Y - X^{1T} Y^1).$$

We do not employ more complicated (deep) learning models, such as TLSTM [12], as our setting requires that model training and scoring need to be highly efficient, as described in Section 3.1.1. The performance of more sophisticated models rely on a large number of training samples [52], which may not be available in the replica write history. For instance, Cassandra regularly empties old commit logs and compacts SSTables. In our comparative evaluation (Section 3.5), we show that our simpler models outperform TLSTM in prediction accuracy.

Generalization Bound

The generalization error of the linear regression model is bounded [98]. This guarantees the prediction accuracy of the estimated $t^1_{n\ \ 1}$ (computed by model f) is bounded compared to the true $t_{n\ \ 1}$. We derive the lower bound for the confidence of replica currency estimation in Propositions 2 and 3.

Proposition 1 ([98])**.** *Let $\hat{R}_Z$pfq denote the empirical loss over the sample training data Z, where |Z| z, and 5 be the bound for the absolute loss function, i.e., |fpt$_{k\ \ 1}$q t$_k$| ¤ 5. For any δ ¡ 0, with probability at least 1 δ over the sample Z, we have*

$$|t^1_{n\ \ 1} \quad t_{n\ \ 1}| \quad \hat{R}_Z\mathsf{p}f\mathsf{q} \quad 5 \quad \frac{\mathsf{c}\ \overline{4\log\frac{ez}{2}}}{z} \quad 5 \quad \frac{\mathsf{d}\ \overline{\log\frac{1}{\delta}}}{2z}.$$

That is, the following probabilistic inequality holds

$$P \quad |t^1_{n\ \ 1} \quad t_{n\ \ 1}| \quad \gamma \quad ¡ \ 1 \quad \delta,$$

where

$$\gamma \quad \hat{R}_Z\mathsf{p}f\mathsf{q} \quad 5 \quad \frac{\mathsf{c}\ \overline{4\log\frac{ez}{2}}}{z} \quad 5 \quad \frac{\mathsf{d}\ \overline{\log\frac{1}{\delta}}}{2z}. \tag{3.2.5}$$

3.2.2 Confidence Bounds for Replica Currency

Based on the predicted $t^1_{n\ \ 1}$ in Formula (3.2.4) and timestamp T_n of the last retrieved version v_n, we estimate the timestamp of the next version $v_{n\ \ 1}$, i.e., $T^1_{n\ \ 1} \quad T_n \quad t^1_{n\ \ 1}$.

Referring to Formulas (3.2.1) and (3.2.2), if $T^1_{n\ \ 1}$ ¤ ζ, we estimate that v_n is stale; otherwise, v_n is current.

In this section, we derive a theoretical bound on the confidence to estimate whether a replica is current or stale. Specifically, what is the probability of the true $T_{n\ \ 1}$ ¤ ζ, given the stale estimation $T^1_{n\ \ 1}$ ¤ ζ? Similarly, for currency estimation, $T^1_{n\ \ 1}$ ¡ ζ, what is the confidence that the true $T_{n\ \ 1}$ ¡ ζ?

Confidence of Currency Estimation

As discussed in Section 3.2.1, the replica currency of v_n is determined by the next version $v_{n\ \ 1}$. If $\zeta \quad T_{n\ \ 1}$, i.e., the next replica version $v_{n\ \ 1}$ occurs after the current query time, we conclude the retrieved version v_n is current.

From Formula (3.2.2), we denote the confidence of a replica version v_n being current by

$$\begin{aligned} P\mathfrak{p}v_n \text{ is current}\mathfrak{q} \quad & P\mathfrak{p}\zeta \quad T_{n\ \ 1}\mathfrak{q} \\ & P\mathfrak{p}\zeta \quad T_n \quad t_{n\ \ 1}\mathfrak{q}. \end{aligned} \tag{3.2.6}$$

Based on the generalization bound between $t_{n\ \ 1}$ and the predicted $t^1_{n\ \ 1}$ in Proposition 1, we derive the lower bound on the confidence $P\mathfrak{p}\zeta \quad T_{n\ \ 1}\mathfrak{q}$, given the estimation $\zeta \quad T^1_{n\ \ 1} \quad T_n \quad t^1_{n\ \ 1}$. For brevity, we provide proof sketches. All complete proofs can be found in the full version technical report [92].

Proposition 2. *To estimate a current replica,*

$$\zeta \quad T^{1}_{n \quad 1},$$

we have the estimation confidence as

$$P\mathfrak{p}v_n \text{ is current}\mathfrak{q} \quad P\mathfrak{p}\zeta \quad T_{n \quad 1}\mathfrak{q} \;\mathfrak{i}\; 1 \quad \delta,$$

where

$$\delta \quad e^{\;2z\mathfrak{p}\frac{T^{1}_{n \quad 1} \quad \zeta \quad \hat{R}_{Z}\mathfrak{p}f\mathfrak{q}}{5}} \quad \mathfrak{b}\frac{}{\frac{4\log\frac{ez}{2}}{z}}\mathfrak{q}^{2}.$$

Proof Sketch [92]. As shown in Figure 3.4, the timestamp $T_{n \quad 1}$ is in the range of $\mathfrak{p}T^{1}_{n \quad 1} \quad \gamma, T^{1}_{n \quad 1} \quad \gamma\mathfrak{q}$ with probability at least $1 \quad \delta$. If the query time has $\zeta \quad T^{1}_{n \quad 1} \quad \gamma$, as illustrated in Figure 3.4(a), we have $\zeta \quad T^{1}_{n \quad 1} \quad \gamma \quad T_{n \quad 1}$, i.e., the version v_n is current as defined in Formula (3.2.2). □

Confidence of Stale Estimation

We now estimate the probability that v_n is stale, i.e., if $T_{n \quad 1} \quad \zeta$, the next version $v_{n \quad 1}$ will occur before the query time ζ, but the replica version has not yet been retrieved.

Referring to Formula (3.2.1), the corresponding confidence of the replica version v_n being stale is

$$P\mathfrak{p}v_n \text{ is stale}\mathfrak{q} \quad P\mathfrak{p}T_{n\ \ 1} \text{ ¤ } \zeta\mathfrak{q} \tag{3.2.7}$$
$$P\mathfrak{p}T_n \quad t_{n\ \ 1} \text{ ¤ } \zeta\mathfrak{q}.$$

Similarly, the lower bound for the confidence $P\mathfrak{p}T_{n\ \ 1} \text{ ¤ } \zeta\mathfrak{q}$, given the estimation for a stale replica $T^1_{n\ \ 1} \text{ ¤ } \zeta$, is guaranteed by studying the generalization error on $t^1_{n\ \ 1}$ and $t_{n\ \ 1}$ in Proposition 1.

Proposition 3. *To estimate a stale replica,*

$$T^1_{n\ \ 1} \text{ ¤ } \zeta,$$

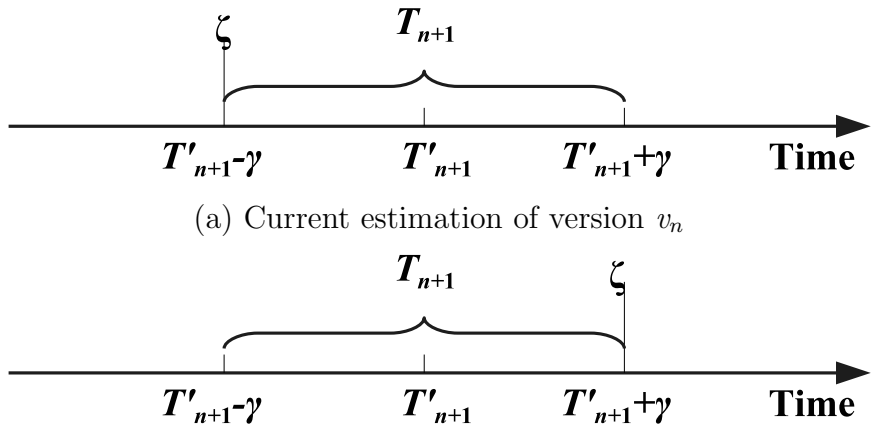
we have the estimation confidence as

$$P\mathfrak{p}v_n \text{ is stale}\mathfrak{q} \quad P\mathfrak{p}T_{n\ \ 1} \text{ ¤ } \zeta\mathfrak{q} \text{ ¡ } 1 \quad \delta,$$

where

$$\delta \quad e^{2z\mathfrak{p}\frac{T^1_{n\ \ 1}\ \ \zeta\ \ \hat{R}_Z\mathfrak{p}f\mathfrak{q}}{5}} \quad \mathfrak{b}\frac{}{\frac{4\log\frac{\epsilon z}{2}}{z}}\mathfrak{q}^2.$$

Proof Sketch [92]. Similar to the proof of Proposition 2, in Figure 3.4(b), if the query time is $\zeta \quad T^1_{n\ \ 1} \quad \gamma$, we have $T_{n\ \ 1} \text{ ¤ } \zeta \quad T^1_{n\ \ 1} \quad \gamma$, i.e., the version v_n is stale as defined in Formula (3.2.1). □



(a) Current estimation of version v_n

(b) Stale estimation of version v_n

Figure 3.4: Replica currency estimation with model generalization bound w.r.t. γ in Formula (3.2.5).

3.3 Realistic Scenarios

Assuming the complete update history exists in a single node for model training and scoring is expensive and unrealistic in our settings with heterogeneous clusters of varying response times, node availability, and writes not being propagated to all nodes. In this section, we consider more realistic scenarios where models are learned locally at each replica node, and together with the data, are sent to the coordinator node for query processing and currency estimation. We extend the theoretical bound for the replica currency estimation confidence (derived under a centralized environment in Section 3.2.2) to distributed scenarios. This facilitates implementation of replica currency estimation in industrial-strength distributed database systems (Section 3.4).

In Section 3.3.1, we describe the realistic settings of replica currency estimation in a distributed environment. We then present our solution that trains individual

prediction models for each replica, and compute the corresponding theoretical bounds for the confidence estimation. Lastly, we assemble the currency estimation results across all the unseen replicas in Section 3.3.3.

3.3.1 Replica Currency

In a distributed database system, the communication costs among different nodes highly variable and sensitive to data partitioning, query workload requests, and node reliability. Data exchanges between replicas should be minimized as much as possible. The replicas are thus asynchronous, i.e., a specific replica version may only exist in a subset of replicas in the distributed system. For example, as shown in Figure 3.5, version v_{44} is written only in replica r_4. Synchronization among the replicas should be conducted minimally, e.g., regularly by anti-entropy or occasionally by read repair, as in Cassandra. For instance, at time T_{42} in Figure 3.5, versions are synchronized in all replicas, i.e., v_{42} and v_{22} are the same version stored in replicas r_4 and r_2, respectively.

In such a real distributed setting, propagating the full history of all writes among nodes, as in Figure 3.2, is unlikely. Therefore, for a specific replica, we can only learn a local model based on its own replica versions. Our goal is to introduce as little overhead as possible, where the learned models are incidentally shared with other replicas during regular and irregular synchronization. For instance, in Figure 3.5, the model f_2 learned locally in replica r_2 is propagated to r_4 when performing a read repair at time T_{42}.

Once the latest synchronization is complete, subsequent writes are not visible

between replicas. For example, in Figure 3.5, the version v_{23} written in r_2 after timestamp T_{42} cannot be seen by replica r_4. In addition, we do not assume in this scenario that all versions before the latest version are available for model training and scoring, i.e., v_5 in Figure 3.2. This relaxation introduces new challenges to estimate the currency of the retrieved version v_{44} in Figure 3.5.

Let v_{pn} with timestamp T_{pn} be the most recent version of the queried data item in the response nodes, and ζ be the query time. The replica's currency in this realistic scenario is dependent on the invisible versions v_{qm} with timestamp T_{qm}, in a replica r_q that has not yet been retrieved. If there exists a version v_{qm} having

$$T_{pn} \quad T_{qm} \text{ ¤ } \zeta, \tag{3.3.1}$$

then version v_{pn} is stale, i.e., a more recent version v_{qm} is written to the other replicas but not retrieved yet. Otherwise, with

$$T_{q,m} \;_1 \text{ ¤ } T_{pn}, \text{ and } \zeta \quad T_{qm}, \tag{3.3.2}$$

we can conclude that v_{pn} is current. In this case, no writes have occurred between the latest version v_{pn} and the query time ζ, and the next write operation will occur beyond the current query time.

Example 7: Figure 3.5 shows v_{44} is the latest retrieved version with timestamp T_{44}, with a response by replica r_4. If there is a version v_{24} written in replica r_2 with timestamp $T_{44} \quad T_{24} \quad \zeta$, and replica r_2 has not responded yet, as illustrated in Figure 3.5(a), then v_{44} is stale. In contrast, if there is no version (across all the

invisible replicas) between T_{44} and ζ, i.e., the next version v_{24} has not yet occurred, then the retrieved version v_{44} is current. ∎

3.3.2 Estimating Single Invisible Replica

To estimate the currency of the observed replica version v_{pn}, similar to the ideal scenario, we need to predict the timestamp T_{qm} of unseen version v_{qm}, namely T^1_{qm}. Specifically, we predict t^1_{qm}, for each invisible replica r_q that has not yet responded to the query. Let f_q be the model in Formula (3.2.3) that is learned locally over the written versions in replica r_q. According to Formula (3.2.4), we have

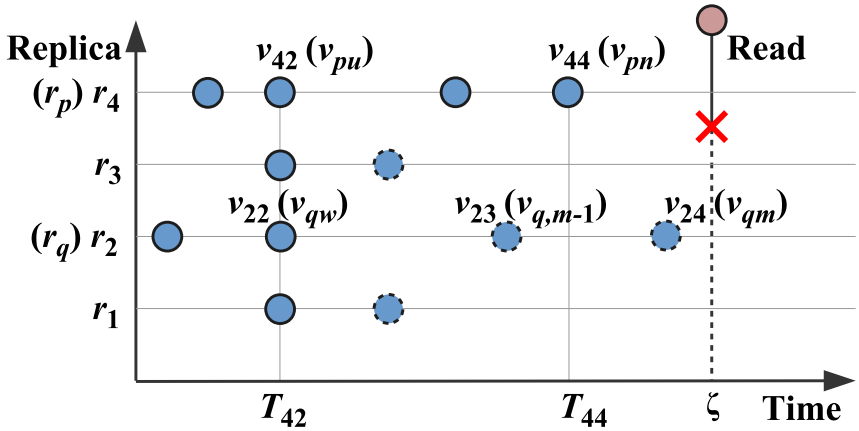
$$t^1_{q,k} \quad 1 \quad \phi_{q1} t_{qk} \quad \phi_{q0}, \tag{3.3.3}$$

where v_{qk} denotes the k-th version of the q-th replica r_q (that has not yet responded to the query), and ϕ_{q0}, ϕ_{q1} denote the model parameters. Referring to Proposition 1 with

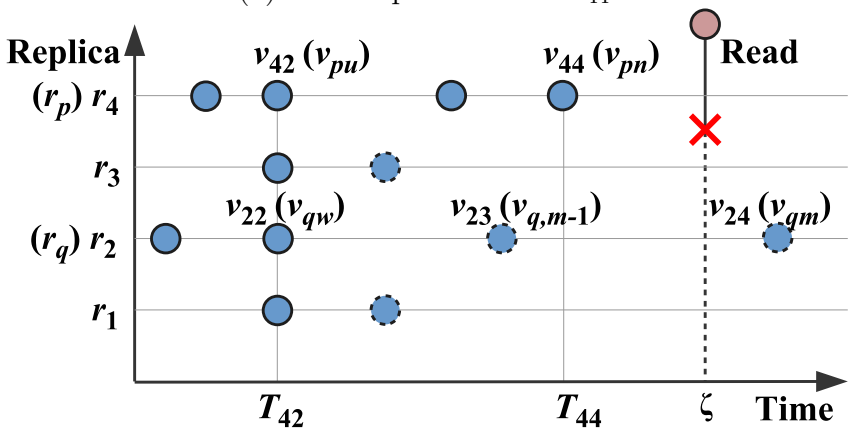
$$\gamma_q \quad \hat{R}_{Z_q} \mathsf{p} f_q \mathsf{q} \quad 5 \quad \mathsf{d} \frac{4 \log \frac{e z_q}{2}}{z_q} \quad 5 \quad \mathsf{d} \frac{\log \frac{1}{\delta}}{2 z_q},$$

the generalization error between t^1_{qk} and t_{qk} is bounded by

$$P \mathsf{p} t^1_{qk} \quad \gamma_q \quad t_{qk} \quad t^1_{qk} \quad \gamma_q \mathsf{q} \ \mathsf{i} \ 1 \quad \delta. \tag{3.3.4}$$



(a) Stale replica version v_{44}



(b) Current replica version v_{44}

Figure 3.5: Replica currency estimation in a realistic scenario, where versions since the last synchronization are not available nor shared, such as at T_{42}.

Predicting Timestamps for Currency Estimation

There may exist multiple versions of r_q that are invisible to the responding replica r_p, after the last synchronization at time T_{qw} with version number v_{qw}, due to a read repair request or anti-entropy.

To predict T_{qm}, or equivalently t_{qm}, we iteratively apply the prediction in Formula (3.3.3), starting from the known version v_{qw} of the last synchronization. The predicted time distance t^1_{qm} of replica version v_{qm} to $v_{q,m\ \ 1}$ has

$$t^1_{qm} \quad \mathsf{p}\phi_{q1}\mathsf{q}^{m\ \ w} t_{qw} \quad \phi_{q0} \sum_{l\ \ 0}^{m\ \ w\ \ 1} \mathsf{p}\phi_{q1}\mathsf{q}^{l}. \tag{3.3.5}$$

Given $T^1_{qm} \quad T^1_{q,m\ \ 1} \quad t^1_{qm}$ for the predicted timestamp of replica version v_{qm}, it follows that

$$\begin{aligned} T^1_{qm} \quad & T_{qw} \sum_{k\ \ w\ \ 1}^{m} t^1_{qk} \\ & T_{qw} \sum_{k\ \ 2}^{m\ \ w\ \ 1} \mathsf{pp}\phi_{q1}\mathsf{q}^{k\ \ 1} t_{qw} \quad \phi_{q0} \sum_{l\ \ 0}^{k\ \ 2} \mathsf{p}\phi_{q1}\mathsf{q}^{l}\mathsf{q}. \end{aligned} \tag{3.3.6}$$

Referring to Formula (3.3.1), if the predicted time stamp T^1_{qm}, $T_{pn} \quad T^1_{qm}$ ¤ ζ, the observed version v_{pn} is estimated to be stale. Otherwise, as in Formula (3.3.2), with $T^1_{q,m\ \ 1}$ ¤ T_{pn} and $\zeta \quad T_{qm}$, we can estimate that v_{pn} is current. The confidence of this currency estimation is the probability of $T_{pn} \quad T_{qm}$ ¤ ζ, given the stale estimation $T_{pn} \quad T^1_{qm}$ ¤ ζ, and similarly for the current estimation.

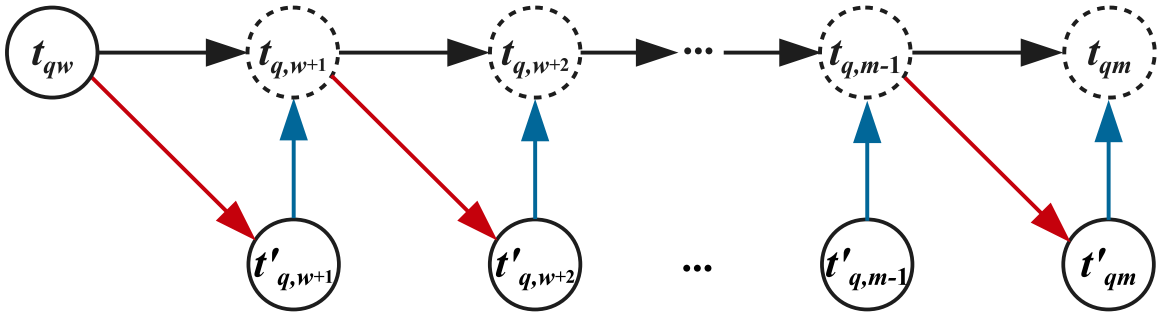


Figure 3.6: Predicting multiple, unseen replica versions. The predictions (shown as red arrows) are introduced in Formula (3.3.3), and the computed bounds (shown as blue arrows) are guaranteed by Formula (3.3.4).

Generalization bound for Timestamp Prediction

Our next goal is to compute the bound on the confidence, e.g., $P\mathfrak{p}T_{pn} \quad T_{qm} \text{ ¤ } \zeta\mathfrak{q}$ of the stale estimation $T_{pn} \quad T^1_{qm} \text{ ¤ } \zeta$ (Section 3.3.2). To do so, we first derive bounds for T_{qm} w.r.t. the timestamp T_{qw} of the last known synchronization, by iteratively applying the bounds in Formula (3.3.4). We combine the relationships between T^1_{qm} and T_{qw} in Formula (3.3.6), to get the bounds on T_{qm} and T^1_{qm} below, which we use to derive the confidence bounds in Propositions 4 and 5.

Lemma 1: *Let $\hat{R}_{Z_q}\mathfrak{p}f_q\mathfrak{q}$ denote the empirical loss over sample training data Z_q in replica r_q, where $|Z_q| \quad z_q$, and 5 be the bound for absolute loss function, i.e., $|f_q\mathfrak{p}t_{q,k} \quad {}_1\mathfrak{q} \quad t_{q,k}| \text{ ¤ } 5$. For any δ ¡ 0, the following probabilistic inequality holds*

$$P\mathfrak{p}T^1_{qm} \quad \beta_{qm}\gamma_q \quad T_{qm} \quad T^1_{qm} \quad \beta_{qm}\gamma_q\mathfrak{q} \text{ ¡ } \mathfrak{p}1 \quad \delta\mathfrak{q}^{\frac{\mathfrak{p}m \quad w\mathfrak{q}\mathfrak{p}m \quad w \quad 1\mathfrak{q}}{2}}, \tag{3.3.7}$$

where

$$\beta_{qm} \quad \sum_{k \quad 2}^{m \quad w} \sum_{l \quad 0}^{1 \, k \quad 2} \mathsf{p}\phi_{q1}\mathsf{q}^{l} \tag{3.3.8}$$

and

$$\gamma_q \quad \hat{R}_{Z_q}\mathsf{p}f_q\mathsf{q} \quad \mathsf{5} \quad \mathsf{d}\frac{4\log\frac{ez_q}{2}}{z_q} \quad \mathsf{5} \quad \mathsf{d}\frac{\log\frac{1}{\delta}}{2z_q}. \tag{3.3.9}$$

I

Proof Sketch [92]. As shown in Figure 3.6, by combining Formulas (3.3.3) and (3.3.4), we have the generalization error bound for t^1_{qm}. The bound for T^1_{qm} is thus guaranteed by considering the error bound for all the previous time difference predictions t^1_{qk}, $w \quad k$ ¤ m. □

Confidence of Currency Estimation

As shown in Section 3.3.1, if $T_{q,m \quad 1}$ ¤ T_{pn} and $\zeta \quad T_{qm}$, i.e., no writes occur between the latest version v_{pn} and the query time ζ, we consider the retrieved version v_{pn} to be current. According to Formula (3.3.2), we have the confidence of a replica version v_{pn} being current as,

$$P\mathsf{p}v_{pn} \text{ is current}\mathsf{q} \quad P\mathsf{p}T_{q,m \quad 1} \text{ ¤ } T_{pn}\mathsf{q}P\mathsf{p}\zeta \quad T_{qm}\mathsf{q}. \tag{3.3.10}$$

Referring to the generalization bound between T_{qm} and the predicted value T^{1}_{qm} in Lemma 1, we derive the lower bound for the confidence $P\mathfrak{p}T_{q,m\ \ 1}\ \mathfrak{\text{¤}}\ T_{pn}\mathfrak{q}P\mathfrak{p}\zeta\quad T_{qm}\mathfrak{q}$, given the current estimation $T^{1}_{q,m\ \ 1}\ \mathfrak{\text{¤}}\ T_{pn}$ and $\zeta\quad T^{1}_{qm}$.

Proposition 4. *For a currency replica estimation on v_{pn} having*

$$T^{1}_{q,m\ \ 1}\ \mathfrak{\text{¤}}\ T_{pn} \text{ and } \zeta\quad T^{1}_{qm},$$

the corresponding estimation confidence is bounded by

$$\begin{aligned} P\mathfrak{p}v_{pn} \text{ is current}\mathfrak{q}\quad & P\mathfrak{p}T_{q,m\ \ 1}\ \mathfrak{\text{¤}}\ T_{pn}\mathfrak{q}P\mathfrak{p}\zeta\quad T_{qm}\mathfrak{q} \\ & \mathfrak{\text{¡}}\,\mathfrak{p}1\quad \delta_{m\ \ 1}\mathfrak{q}^{\frac{\mathfrak{p}m\ \ w\ \ 1\mathfrak{q}\mathfrak{p}m\ \ w\mathfrak{q}}{2}}\mathfrak{p}1\quad \delta_{\zeta}\mathfrak{q}^{\frac{\mathfrak{p}m\ \ w\mathfrak{q}\mathfrak{p}m\ \ w\ \ 1\mathfrak{q}}{2}}, \end{aligned}$$

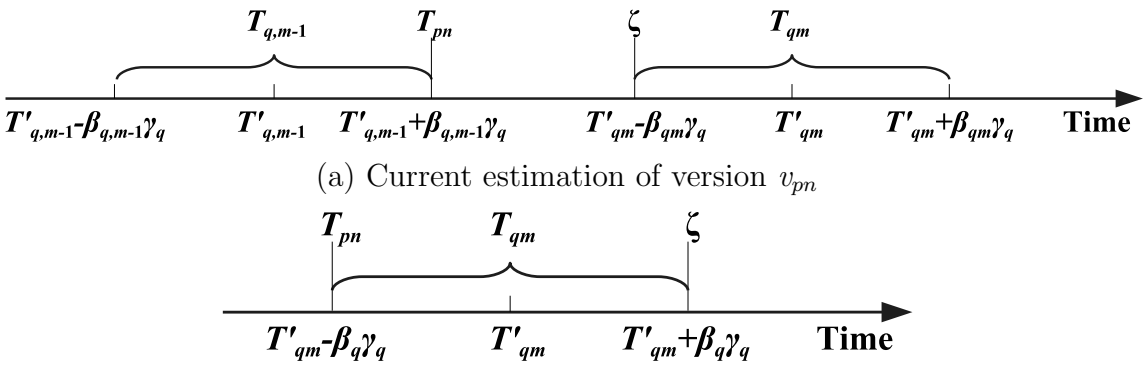
where

$$\delta_{m\ \ 1}\quad e^{2z_q\mathfrak{p}\frac{T_{pn}\ \ T^{1}_{q,m\ \ 1}\ \ \beta_{q,m\ \ 1}\hat{R}_{Z_q}\mathfrak{p}f_q\mathfrak{q}}{5\beta_{q,m\ \ 1}}\ \ \mathfrak{c}\frac{\sqrt{4\log\frac{ez_q}{2}}}{z_q}\mathfrak{q}^{2}}$$

and

$$\delta_{\zeta}\quad e^{2z_q\mathfrak{p}\frac{T^{1}_{qm}\ \ \zeta\ \ \beta_{qm}\hat{R}_{Z_q}\mathfrak{p}f_q\mathfrak{q}}{5\beta_{qm}}\ \ \mathfrak{c}\frac{\sqrt{4\log\frac{ez_q}{2}}}{z_q}\mathfrak{q}^{2}}.$$

Proof Sketch [92]. According to Lemma 1, the timestamps T_{qm} and $T_{q,m\ \ 1}$ can be theoretically bounded, as shown in Figure 3.7. If the query time ζ has $\zeta\quad T^{1}_{qm}\quad \beta_{qm}\gamma_q$, as illustrated in Figure 3.7(a), we have $\zeta\quad T^{1}_{qm}$. Moreover, if the retrieved replica version v_{pn} has $T_{pn}\quad T^{1}_{q,m\ \ 1}\quad \beta_{q,m\ \ 1}\gamma_q$, we have $T_{pn}\ \mathfrak{\text{¥}}\ T_{q,m\ \ 1}$. □



(a) Current estimation of version v_{pn}

(b) Stale estimation of version v_{pn}

Figure 3.7: Replica currency estimation with model generalization bound w.r.t. β_q (Formula (3.3.8)), and γ_q (Formula (3.3.9)).

Confidence of Stale Estimation

Recall that if there exists at least one replica version v_{qm} occurring after the latest retrieved v_{pn}, and before the query time ζ, i.e., T_{pn} T_{qm} ¤ ζ, then the retrieved replica version v_{pn} is stale. Assume that the time T_{pn} and T_{qm} are independent of the query read time ζ. Referring to Formula (3.3.1), the corresponding confidence of the replica version r_{pn} being stale is

$$P\mathfrak{p}v_{pn} \text{ is stale}\mathfrak{q} \quad P\mathfrak{p}T_{pn} \quad T_{qm} \text{ ¤ } \zeta\mathfrak{q}$$
$$P\mathfrak{p}T_{pn} \quad T_{qm}\mathfrak{q}P\mathfrak{p}T_{qm} \text{ ¤ } \zeta\mathfrak{q}.$$

Similarly, the lower bounds of the confidence $P\mathfrak{p}T_{pn}$ T_{qm} ¤ $\zeta\mathfrak{q}$, given a stale replica version estimation T_{pn} T^1_{qm} ¤ ζ, are guaranteed by studying the generalization error on T^1_{qm} and T_{qm} in Lemma 1.

Proposition 5. *For a stale replica estimation on v_{pn} having*

$$T_{pn} \quad T^{1}_{qm} \text{ ¤ } \zeta,$$

the corresponding estimation confidence is bounded by

$$P\mathsf{p}v_{pn} \text{ is stale}\mathsf{q} \quad P\mathsf{p}T_{pn} \quad T_{qm} \text{ ¤ } \zeta\mathsf{q}$$
$$\text{¡ p1} \quad \delta_m\mathsf{q}^{\frac{\mathsf{p}m \quad w\mathsf{q}\mathsf{p}m \quad w \quad 1\mathsf{q}}{2}}\mathsf{p1} \quad \delta_\zeta\mathsf{q}^{\frac{\mathsf{p}m \quad w\mathsf{q}\mathsf{p}m \quad w \quad 1\mathsf{q}}{2}}, \tag{3.3.11}$$

where

$$\delta_m \quad e^{2z_q\mathsf{p}\frac{T^{1}_{qm} \quad T_{pn} \quad \beta_{qm}\hat{R}_{Z_q}\mathsf{p}f_q\mathsf{q}}{5\beta_{qm}}} \quad \mathsf{c}\frac{}{\frac{4\log\frac{ez_q}{2}}{z_q}}\mathsf{q}^2$$

and

$$\delta_\zeta \quad e^{2z_q\mathsf{p}\frac{\zeta \quad T^{1}_{qm} \quad \beta_{qm}\hat{R}_{Z_q}\mathsf{p}f_q\mathsf{q}}{5\beta_{qm}}} \quad \mathsf{c}\frac{}{\frac{4\log\frac{ez_q}{2}}{z_q}}\mathsf{q}^2.$$

Proof Sketch [92]. Similar to the proof of Proposition 4, if the query time ζ has $\zeta \quad T^{1}_{qm} \quad \beta_{qm}\gamma_q$, as illustrated in Figure 3.7(b), we have ζ ¥ T_{qm}. For the retrieved version v_{pn}, if its timestamp T_{pn} has $T_{pn} \quad T^{1}_{qm} \quad \beta_{qm}\gamma_q$, we have $T_{pn} \quad T_{qm}$. □

We note that there may exist multiple versions v_{qm}, having $T_{pn} \quad T^{1}_{qm}$ ¤ ζ, all of which estimate the version v_{pn} being stale. We assume the events of multiple versions v_{qm} occurring in the time range $\mathsf{p}v_{qm}, \zeta\mathsf{s}$ are independent. The overall (enhanced) confidence of the stale replica estimation on v_{pn} is thus obtained by aggregating the

confidences of all v_{qm},

$$P\mathsf{p}v_{pn} \text{ is stale}\mathsf{q} \tag{3.3.12}$$
$$1 \underset{m\mathsf{P}\mathsf{t}m|T_{pn} \quad T^1_{qm}\,\mathsf{¤}\,\zeta\mathsf{u}}{\overset{\mathsf{1}}{}} \mathsf{p}1 \quad P\mathsf{p}T_{pn} \quad T_{qm}\ \mathsf{¤}\ \zeta\mathsf{qq}.$$

3.3.3 Assembling Multiple Invisible Replicas

Section 3.3.2 studies currency estimation w.r.t. one invisible replica, such as r_2 in Figure 3.5. In real scenarios, there may exist multiple replicas that do not reply, e.g., r_1 and r_3 in addition to r_2 in Figure 3.5. The estimation and confidence from each invisible replica need not be equal, posing questions of how to aggregate and reconcile these values. In this section, we aggregate these (potentially conflicting) currency estimations on the last version v_{pn} retrieved thus far, as well as their corresponding confidence values.

Intuitively, if there exists one invisible replica r_q that estimates the retrieved version v_{pn} is stale, e.g., r_2 in Figure 3.5(a), it is sufficient to conclude a stale estimation. Similar to Formula (3.3.12), the corresponding stale estimation confidence aggregates all the invisible replicas that have a stale estimation on v_{pn},

$$P\mathsf{p}v_{pn} \text{ is stale}\mathsf{q} \tag{3.3.13}$$
$$1 \underset{q\mathsf{P}\mathsf{t}q|r_q \text{ invisible}\mathsf{u}\ m\mathsf{P}\mathsf{t}m|T_{pn} \quad T^1_{qm}\,\mathsf{¤}\,\zeta\mathsf{u}}{\overset{\mathsf{1}}{}\quad\overset{\mathsf{1}}{}} \mathsf{p}1 \quad P\mathsf{p}T_{pn} \quad T_{qm}\ \mathsf{¤}\ \zeta\mathsf{qq}.$$

For currency, only when all the invisible replicas estimate the retrieved version v_{pn}

is current, e.g., in Figure 3.5(b), we then conclude v_{pn} is current. The confidence is also calculated by combining the estimations of all the invisible replicas in Proposition 4,

$$P\mathsf{p}v_{pn} \text{ is current}\mathsf{q} \underset{\mathsf{1}}{\underset{q\mathsf{P}\mathsf{t}q|r_q \text{ invisible}\mathsf{u}}{\prod}} P\mathsf{p}T_{q,m} \;\; {}_1 \; \mathsf{¤} \; T_{pn}\mathsf{q}P\mathsf{p}\zeta \quad T_{qm}\mathsf{q}, \tag{3.3.14}$$

where m corresponds to version v_{qm} for the invisible replica r_q, leading to the current estimation $T^{1}_{q,m} \;\; {}_1 \; \mathsf{¤} \; T_{pn}$ and $\zeta \quad T^{1}_{qm}$.

3.4 Cassandra Implementation

This section presents our implementation in an open-source, distributed database, Apache Cassandra [5]. We believe that our replica currency estimation approaches work similarly in other (NoSQL) databases, such as HBase [6], which are avenues for future work. We implement two major modifications to the Cassandra system: (1) we deploy training models in individual nodes, and propagate them during node synchronization; and (2) we perform replica currency estimation, and enable the novel DYNAMIC consistency level during query processing.

3.4.1 Replica Synchronization

In the realistic scenarios, shown in Figure 3.5, a prediction model f_p is learned locally in each replica node r_p following the incremental learning method in Section 3.2.1.

However, it is not necessary to immediately update the model after each write, which degrades write performance.

Instead, model training can either be scheduled regularly (daily/weekly), possibly together with the regular anti-entropy operations in Cassandra, or irregularly when a read repair on the data item occurs. Given this, our implementation introduces no additional overhead costs for write operations.

The learned model f_p is propagated to all the other replicas of the data item for currency estimation and query processing. Let T_{pu} be the timestamp of a replica synchronization operation e.g., regularly by anti-entropy or occasionally by read repair in Cassandra. The latest version v_{pu} of the data item will be synchronized across all replicas. Let f_{pu} be the latest version of model f_p at time T_{pu}, which is also propagated to all replicas.

When a node is re-synchronized (e.g., recovers from failure or being offline), the process learns and propagates the latest models f_{pu} across all replicas. After the synchronization, each replica r_q carries both the latest version of the data item and the latest prediction models f_{pu} from the other replicas r_p. For instance, at time T_{42} in Figure 3.5, versions are synchronized across all replicas, i.e., v_{42} and v_{22} are the same version stored in replicas r_4 and r_2, respectively. Moreover, in addition to the data values, after incremental learning as new versions arrive, the latest prediction model f_{22} of replica r_2 is also sent to replica r_4, and vice versa.

Our work enables systems to reduce query latency where there is variance among

the replica response times. This occurs in cases such as network performance instability, varying node reliability, and geographic distribution of replicas. We study the latter two cases in our evaluation. For instance, in Section 3.5.3, we deploy servers globally using AWS with replica response times with mean 1.35 ms and variance 0.05 ms. We empirically find that increasing synchronization time intervals, and time from synchronization to user query time, results in an average 2.07% and 2.94% decline in query accuracy, as expected, as more stale data arises. While we currently assume that T_{pu} is fixed according to anti-entropy or via read repair, we intend to study in future work how this synchronization time can be adaptive to the replica response times.

3.4.2 Query Processing

We now introduce the necessary changes to query processing, to support replica currency estimation, as illustrated in Figure 3.1(a) and Figure 3.5. The client connects to any node in the cluster as the coordinator for the read requests. The coordinator then contacts all the replicas of the requested data item. Responding replicas r_p send their latest version v_{pn} of the data item as usual, as well as the additional version v_{pu} (synchronized with v_{qw}), and the models f_{qw} of all the replicas r_q (including v_{pu} of itself) during the last synchronization. Recall that this information is necessary to predict T^1_{qm} for currency estimation in Formula (3.3.6).

Based on the replicas that have replied to the query thus far, the coordinator dynamically determines whether waiting for the remaining replicas is warranted, namely

using the DYNAMIC consistency level. First, it estimates whether the retrieved latest version v_{pn} is stale or current, with bounded confidence as presented in Formulas (3.3.13) and (3.3.14), respectively. Rather than retrieving a fixed number of replicas, in the case of existing protocols (ONE, QUORUM, ALL), the system immediately returns v_{pn} if its currency estimation confidence satisfies a threshold η. Our proposed DYNAMIC consistency level offers more finely tunable confidence (e.g., η 0.99, 0.999, 0.9999, ...) to enable users and applications to evaluate the trade-off between replica currency and query efficiency.

For instance, in Figure 3.5 at query time ζ, replica r_4 responds to the coordinator, by returning the latest version v_{44} (as in the Cassandra implementation), and also v_{42} that is synchronized with other replicas such as v_{22}, and their corresponding models f_{22}. Recall that this information is propagated and stored in all the replicas including r_4 during the last synchronization at time T_{42}. If version v_{44} is estimated to be current, with confidence at least η, say 0.999, it will be returned as the query answer together with the lower bound for the currency estimation confidence.

3.5 Experiments

We implement our models using Apache Cassandra 4.0, and evaluate with the following objectives:

1) We compare our estimation models under the ideal and realistic scenarios, and against a state-of-the-art deep learning model.

2) We evaluate the accuracy and confidence of our estimation models, varying synchronization time intervals from the latest node sync time to the query read time and the next node sync time.
3) We evaluate our models by varying query workload characteristics such as network partition time, node down time, # unavailable nodes, # skewed updates/accesses, and # write replicas.
4) We compare our DYNAMIC read consistency level against the existing consistency levels and show its improved accuracy, lower number of retrieved replicas, and reduced time costs.

Our experimental highlights are: (i) the replica currency estimation is accurate with high confidence; (ii) the DYNAMIC consistency level built on replica currency estimation is effective and efficient.

3.5.1 Experimental Setup

We set up a global data centre topology consisting of 10 Amazon EC2 m5.4xlarge instances, where each server contains 16 CPU cores, 64 GB memory, and 8 GB local storage, running Ubuntu 18.04 LTS. The servers are distributed across 5 regions, namely, Canada, China, Germany, South Africa, and the US.

Datasets. We use five real datasets to model query updates.

Btcusd [85] dataset is from the Bitfinex exchange, containing historical trading data (open price) of the trading pair 'Bitcoin vs. US dollar' at 1 min intervals. There

are 5,167 total writes.

Sensor [26] dataset contains temperature sensor readings on server racks in a data center over a week. Approximately 80% of the data is updated every 20 sec, and the total number of writes (as the query load) is 2,864.

Bike [59] dataset contains bike sharing details in London, UK, with an update frequency of 1 min, with 17,414 total writes.

Mental [78] dataset measures attitudes towards mental health, and the frequency of mental health disorders in the workplace. Updates occur irregularly, with 1,259 total writes.

Bank [7] dataset describes bank transactions for customer bank accounts. Transactions occur irregularly, with 1,294 total writes.

Evaluation. While data items are written in the same sequence as in the datasets, we randomly pose read requests in the query loads. As discussed in Section 3.4, samples of the update history stored in replicas are used to incrementally train the prediction model f. For each read request, currency estimation of the returned replica is a binary classification, i.e., current or stale. We use F1-score [98] as our metric to evaluate the currency estimation.

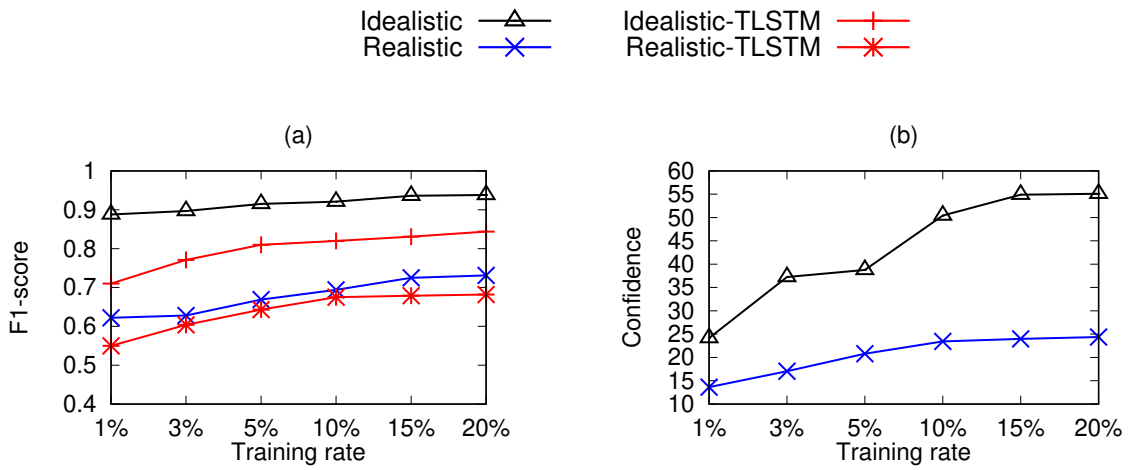


Figure 3.8: Replica currency estimation by varying training rates over Btcusd data.

3.5.2 Comparing Prediction Models

We evaluate the accuracy of our prediction models for replica currency estimation under the ideal and realistic scenarios, described in Sections 3.2 and 3.3, respectively. A state-of-the-art method TLSTM [12] is employed as a competitor to predict the timestamp of the next version.

TLSTM extends the deep learning model LSTM [58] by supporting irregular time intervals. We use its open-source implementation [61]. In addition to the time between adjacent updates, also considered in our proposal, TLSTM uses more information by taking the update values as the additional input.

Figure 3.8 reports the results for varying training rates. With more training data, the accuracy of currency estimation improves. The corresponding confidence of the currency estimations also increases. Figure 3.9 presents the currency estimation performance of our models under various thresholds η. As illustrated in Figure 3.9(b),

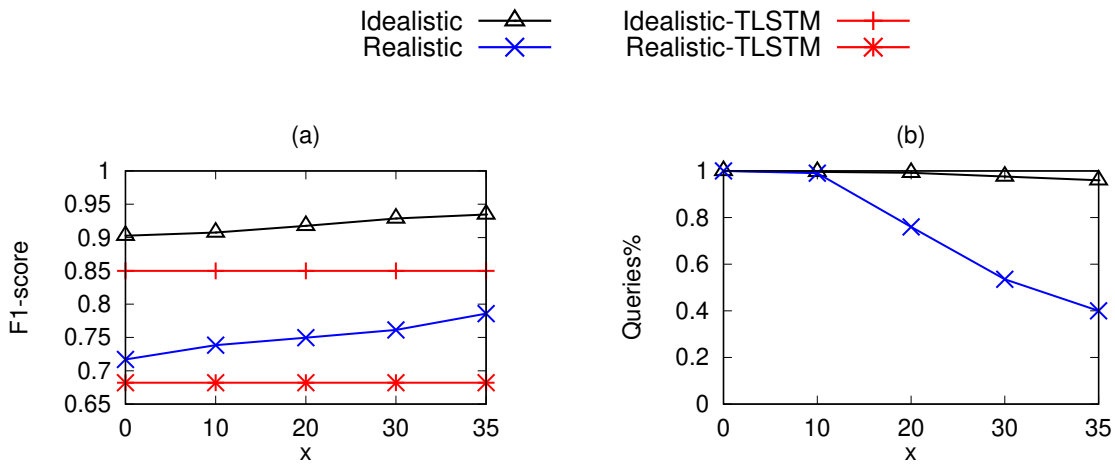


Figure 3.9: Replica currency estimation by varying confidence thresholds $\eta \quad 1 \quad e^{\ x}$ over Btcusd data.

the percentage of all queries reduces with higher requirements of confidence η (or equivalently x). We note that the prediction accuracy of TLSTM, even using more information of update values, is lower than the linear regression used in our proposal. The result is not surprising given that the performance of deep learning models relies on a large number of training samples [52]. It is too expensive to use, as mentioned in Section 3.2.1, since our setting requires the model training and scoring to be highly efficient.

3.5.3 Node Synchronization Evaluation

This experiment studies the node synchronization strategies described in Section 3.4.1, where model training can be scheduled. Figures 3.10 and 3.11 vary the time intervals from the last node synchronization to the next node synchronization, and to the user query time, respectively. According to Propositions 4 and 5, larger intervals

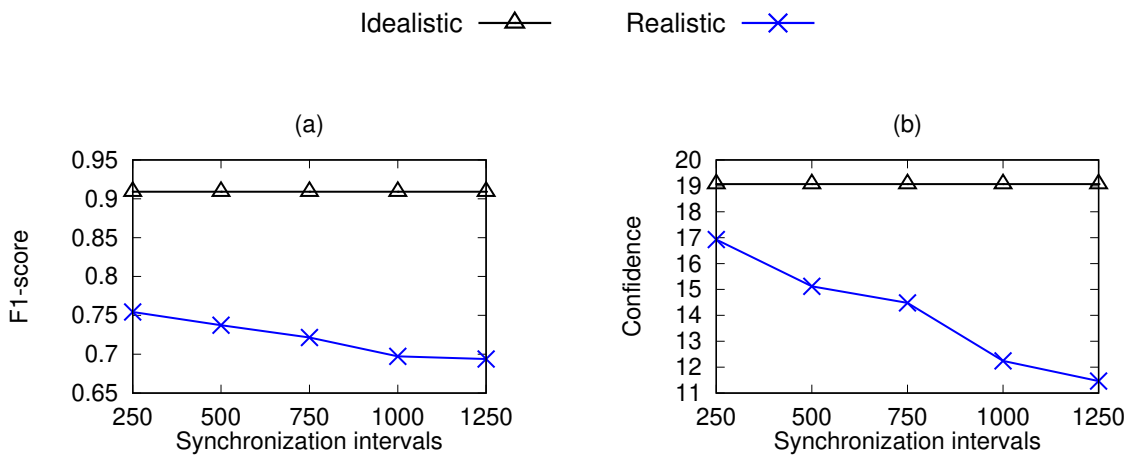


Figure 3.10: Varying time intervals between synchronizations intervals with two consecutive node synchronizations (e.g., T_{42} in Figure 3.5 and the one before it) using Btcusd data.

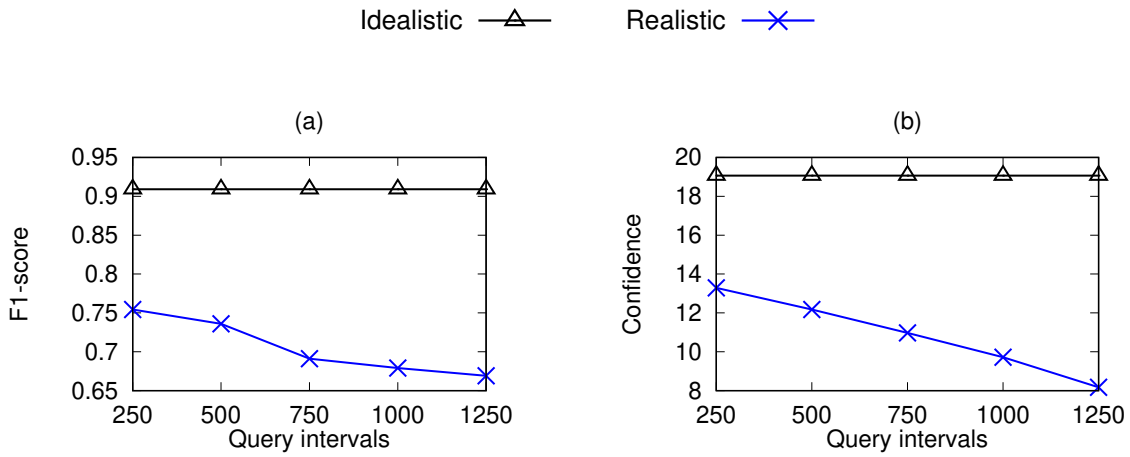


Figure 3.11: Varying time intervals between the latest node synchronization and query time (e.g., T_{42} and ζ in Figure 3.5)

lead to lower confidence for the currency determination. It is also not surprising that the corresponding currency estimation accuracy becomes lower. Figure 3.12(a) shows the time costs for read repair and training under various synchronization time intervals. The corresponding query and estimation scoring time costs are reported

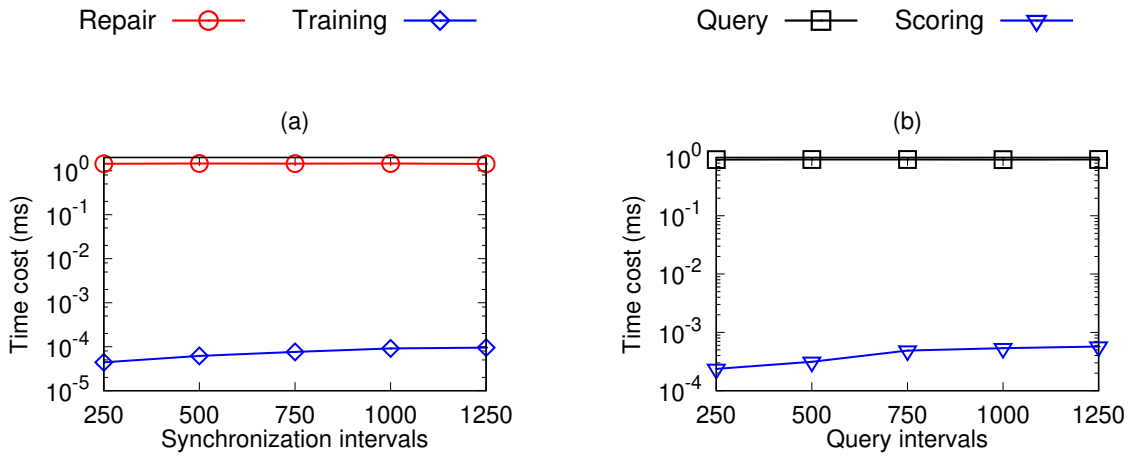


Figure 3.12: Time cost for varying synchronization and query time intervals using Btcusd data

in Figure 3.12(b). Note that the estimation scoring and training time costs are only 0.076%-1.17% of the original costs in query processing and replica synchronization (i.e., during read repair).

3.5.4 Query Processing Evaluation

We evaluate currency estimation in query processing as introduced in Section 3.4.2 under various query loads.

Network Partition and Node Failure

We consider two query load scenarios: (i) network partitioning that specifies clusters to service reads or writes in separate nodes at specific times; and (ii) varying node failure times. Figure 3.13(a) shows the network partitioning scheme, where specific

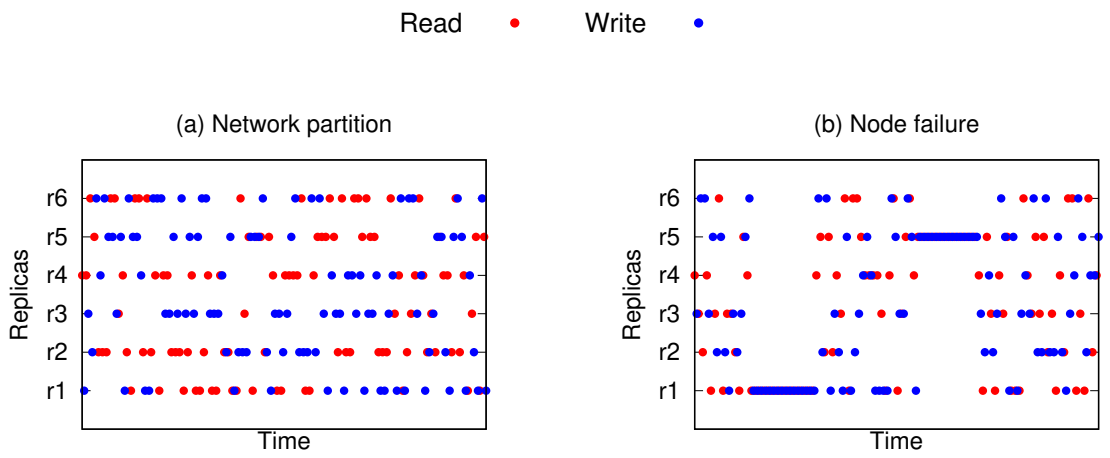


Figure 3.13: Query loads over (a) network partition with separate reads/writes and (b) node failure with suspended service

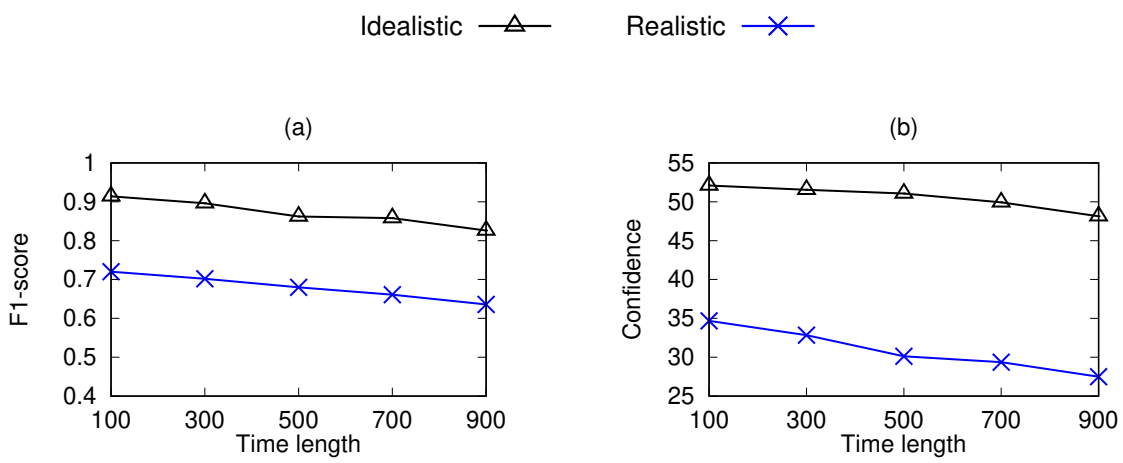


Figure 3.14: Replica currency estimation for varying time lengths of network partition (Figure 3.13(a)) using Btcusd data

replicas service reads or writes exclusively for a specific time interval. Figure 3.13(b) shows the node failure scenario where both reads and writes are not serviced at offline/failure nodes, denoted as empty time intervals. Under the network partitioning case, Figure 3.14 shows that as we vary the time intervals of a network partition, the F1-score and the confidence decrease for longer network partitioning time intervals;

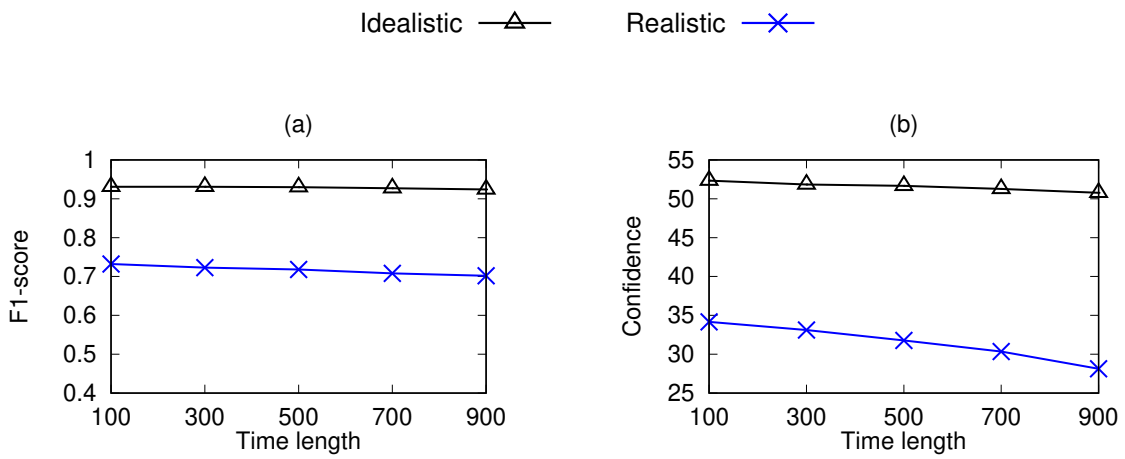


Figure 3.15: Replica currency estimation for varying node failure times (Figure 3.13(b)) using Btcusd data

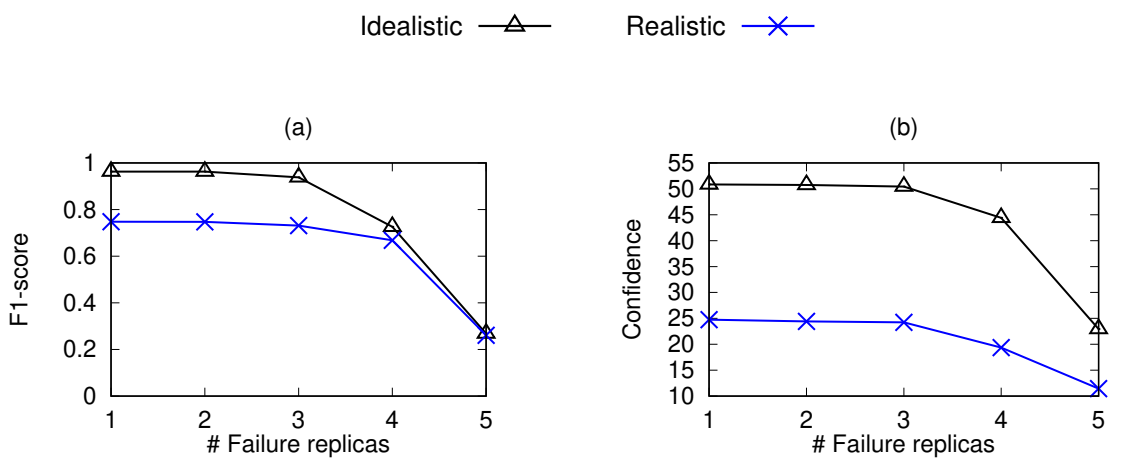


Figure 3.16: Varying number of node failures

making the prediction of the next version more challenging. Figure 3.15 shows a linear decline in F1 accuracy and confidence for increasing node down time. To study the impact of the number of failed nodes on currency estimation, Figure 3.16 shows, as expected, currency estimation becomes more difficult as more nodes become unavailable.

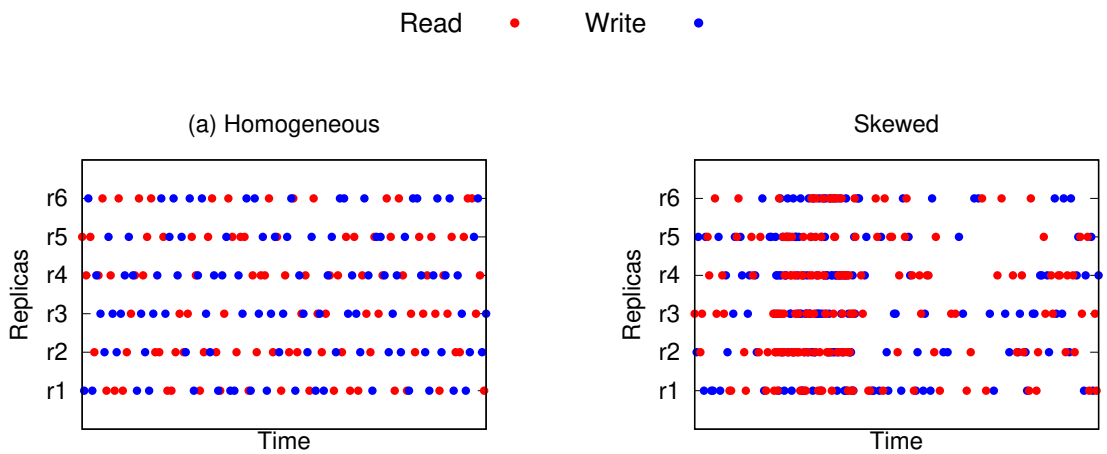


Figure 3.17: Query loads with (a) homogeneous and (b) skewed updates and reads

Skewed Updates and Accesses

We study the impact of varying read/write distributions on our estimation accuracy. Figure 3.17(a) shows homogeneous reads/writes across the replicas. Figure 3.17(b) illustrates skewed reads/writes, which reflect data access patterns varying across time, e.g., more frequent reads/writes during the weekdays vs. the weekends.

Figure 3.18 shows the accuracy and confidence as we vary the skewness parameter S, where a value of $S = 2.7$ indicates the homogeneous case of the original dataset (Figure 3.17(a)). As expected, our estimation accuracy and confidence decline for increasing data skew as it becomes more difficult to learn an accurate model for currency estimation. Our model is adaptable to learn over data with periodic updates (regardless of the time periodicity) (Figure 3.21), with irregular updates (Figure 26 in [92]), and those with increasing data skew (Figure 3.18).

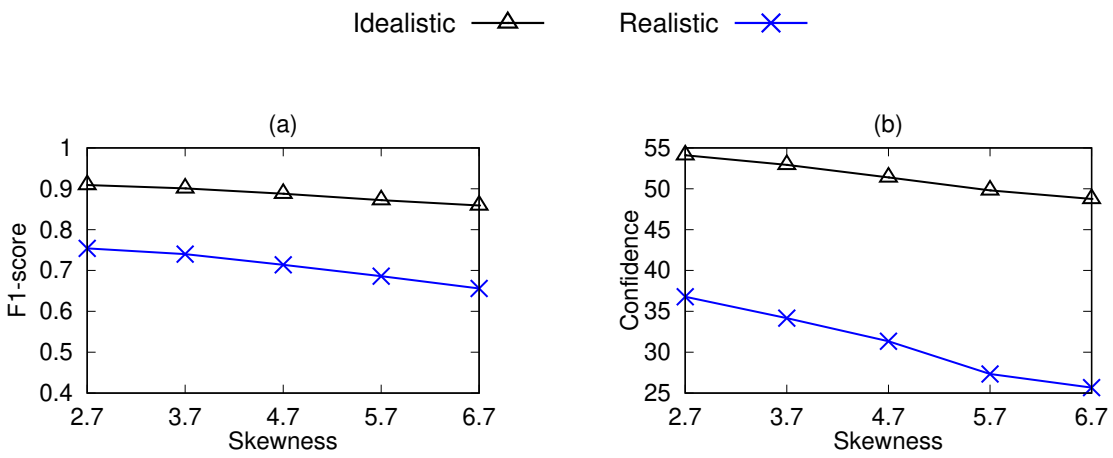


Figure 3.18: Replica currency estimation by varying the skewness of updates and accesses (Figure 3.17(b)) using Btcusd data

Write Consistency Level

We evaluate the impact of varying the number of write replicas, also known as the write consistency level. We simulate the number of write replicas using a Poisson distribution with parameter λ. Figure 3.19(a) shows the distribution of reads/writes for each replica, and Figure 3.19(b) illustrates how writes are Poisson distributed across the number of write replicas. Figure 3.20 reports the results as we vary λ from 0.5 to 2.5, where larger λ leads to more write replicas. The results show that our models are able to learn and adapt to increases in λ, with corresponding increases in F1 score and confidence due to the larger number of write replicas.

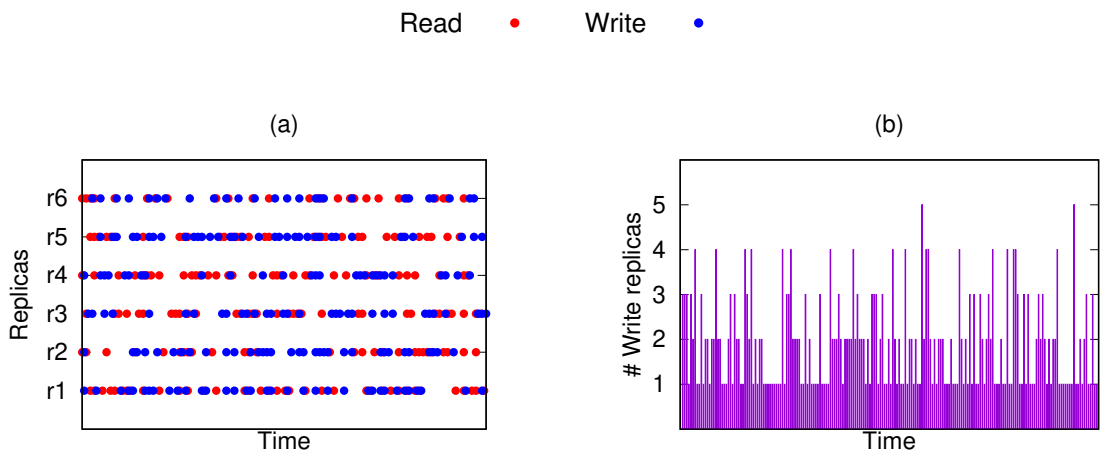


Figure 3.19: Query load by varying write consistency levels (varying number of write replicas)

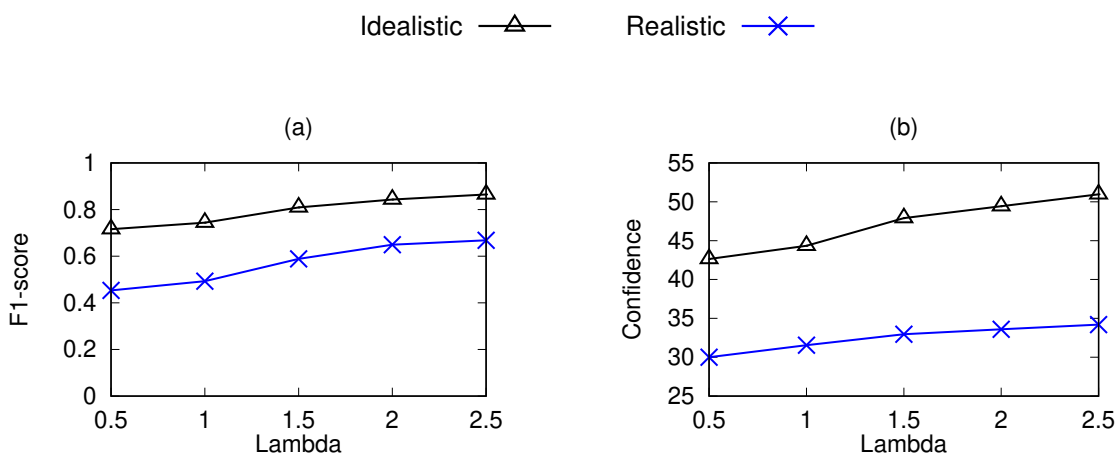


Figure 3.20: Replica currency estimation by varying write consistency levels, according to λ using Btcusd data

3.5.5 DYNAMIC Read Consistency Level

In this experiment, we compare our DYNAMIC read consistency level proposed in Section 3.4 against the existing options (ONE, QUORUM, ALL) with ONE write, and with QUORUM writes. In such a scenario, QUORUM reads with QUORUM writes

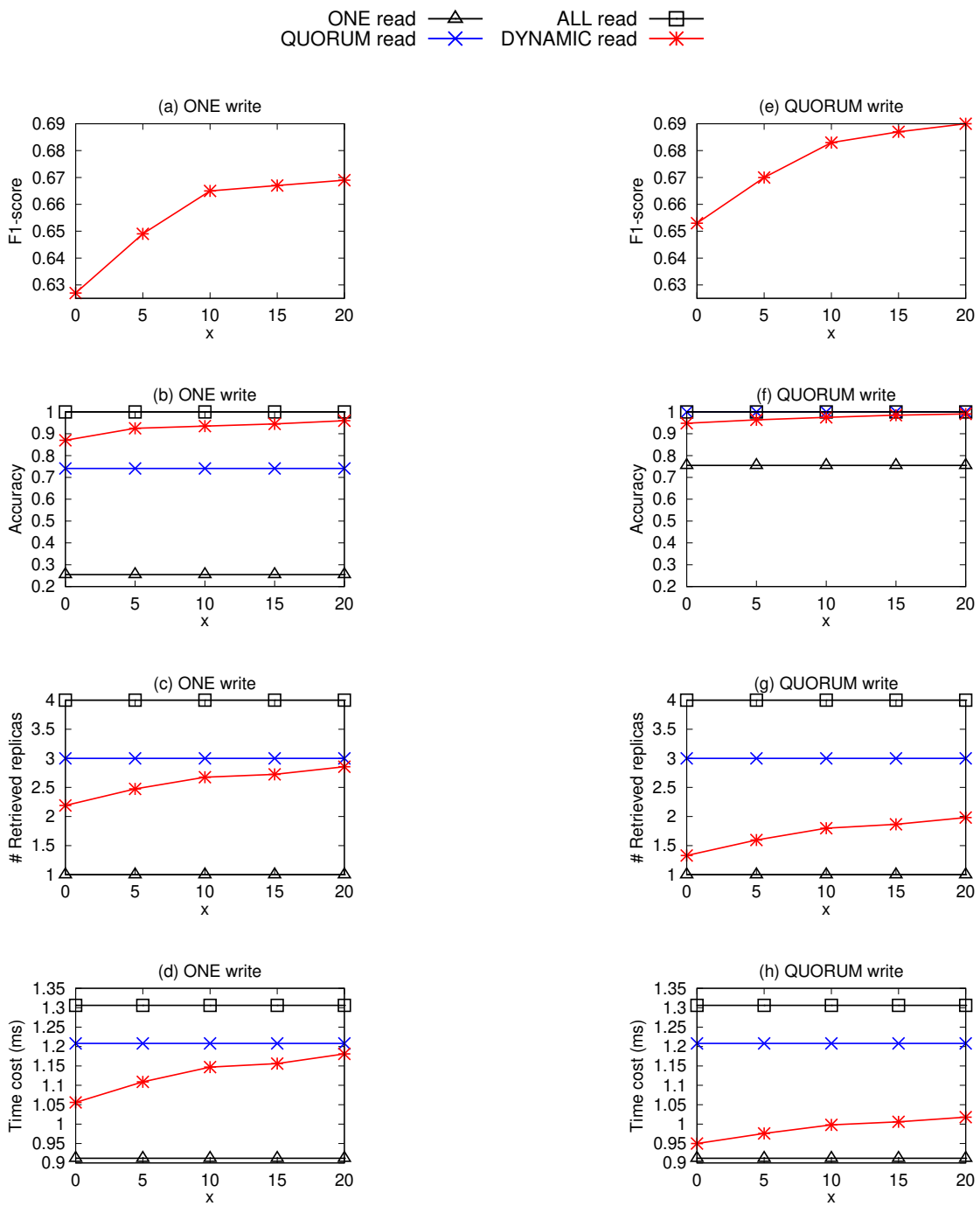


Figure 3.21: DYNAMIC consistency level under various confidence threshold $\eta = 1 - e^{-x}$ using Btcusd data

could guarantee the currency of results, but this is not guaranteed with ONE write. Instead of requiring a fixed number of nodes to respond to a query, the DYNAMIC consistency level immediately returns the response replicas if their confidence of being current satisfies threshold η. Since the existing consistency levels do not consider currency estimation, and all the returned results are assumed to be current, we compare their relative accuracy of being current.

Figures 3.21 and 3.22 show results of our DYNAMIC consistency level over the Btcusd and Bike datasets. Similar results are also observed in the other Sensor, Bank and Mental datasets described in Section 3.5.1, and presented in Figures 24, 25 and 26 in [92]. Figures 3.21(a)-(d) present comparative results using ONE write, followed by QUORUM write in Figures 3.21(e)-(h). As expected, Figures 3.21(a) and 3.21(e) show that as we increase the confidence threshold, the F1 score of currency estimation for each replica increases, with higher accuracy using QUORUM write. In addition, Figure 3.21(b) and 3.21(f) report the accuracy of returning current replicas in query answering, which is guaranteed by ALL-read and QUORUM-read+QUORUM-write. As shown in Figure 3.21(b), the DYNAMIC consistency level achieves a higher accuracy of returning current answers than QUORUM (comparable to ALL) reads. The corresponding number of dynamically retrieved replicas as well as the time costs, however, are less than the fixed QUORUM in Figures 3.21(c) and 3.21(d), respectively. The reason is that our DYNAMIC consistency level immediately returns the answer after retrieving a high confidence current replica, whereas QUORUM reads must still wait for a fixed number of replicas to respond. Such an improvement is more significant with QUORUM write in Figures 3.21(g) and 3.21(h), since there exist more current replicas. The results across all datasets demonstrate that our DYNAMIC

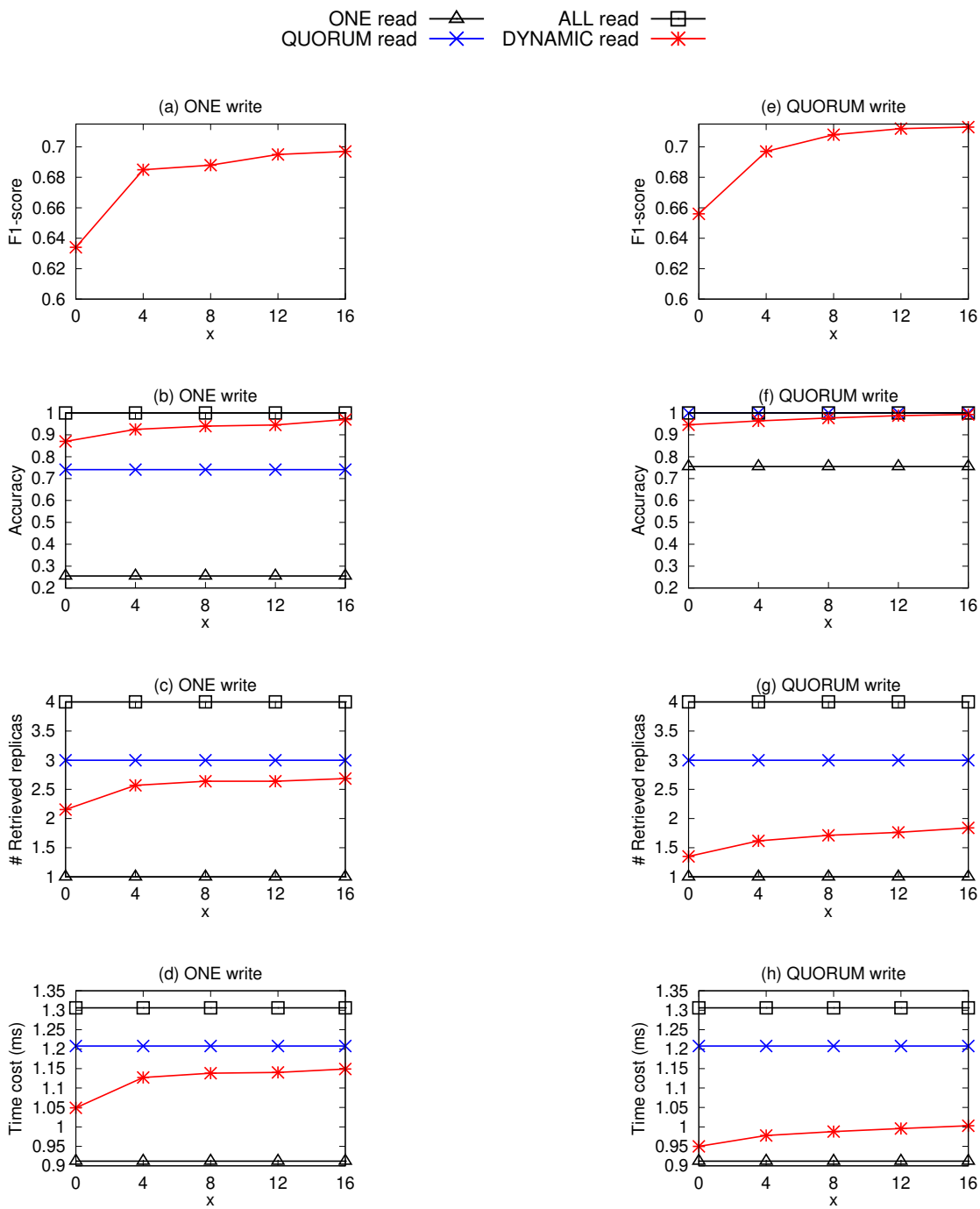


Figure 3.22: DYNAMIC consistency level under various confidence threshold $\eta \quad 1 \quad e^{\ x}$ using Bike data

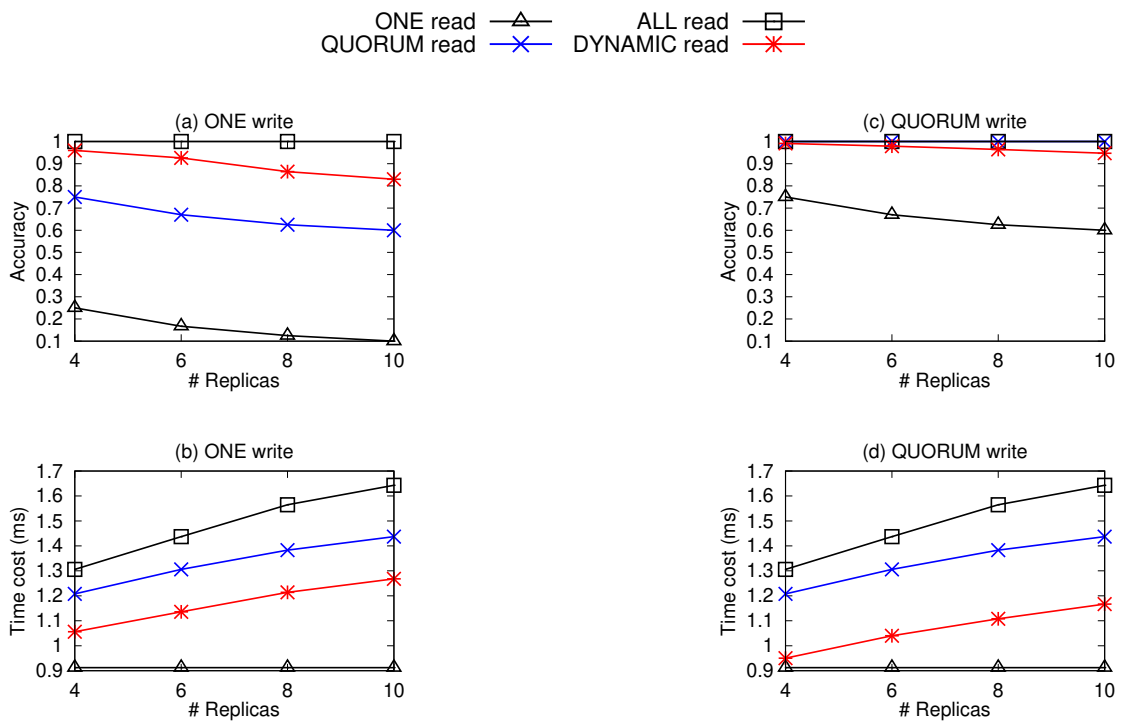


Figure 3.23: DYNAMIC consistency level under various number of servers using Btcusd data

consistency level offers great opportunity for systems to decrease latency costs, and to lower query processing times.

Finally, we consider a real application deployment with varying network performance and replica locations. We use the AWS cloud to geographically distribute cluster nodes across different regions to consider the communication delays corresponding to latencies within a region and across regions. Figure 3.23 shows the results as we vary the numbers of servers located in different regions. It is not surprising that the time cost increases as the number of replicas increases, as shown in Figure 3.23(b) and Figure 3.23(d), which lead to the variance in production latency. Our DYNAMIC consistency level still achieves lower time cost (compared to QUORUM read). At the

same time, more replicas also make the current replica rate smaller under ONE write and QUORUM write, leading to lower accuracy in Figure 3.23(a) and Figure 3.23(c).

3.6 Related Work

3.6.1 Data Currency

Data currency is vital for correct and timely interpretation of data. Fan et al., propose *currency constraints (CCs)* that declaratively define currency relationships according to the domain semantics [39, 40]. CCs are assumed to be given apriori, and elicit a partial currency order among tuples in a relation. For example, tuples t_1 and t_2 describing a person's status as 'working' and 'retired', respectively, indicate that t_1 precedes t_2. CCs have been used to determine currency orders for data values copied between data sources. Based on these copy relationships, CCs facilitate query answering involving only current data [41]. Extensions of this work include reasoning about CCs and their consistency, computation of certain answers regardless of the partial currency order, and studying whether copy functions can import sufficient data to answer a query [44]. CCs, however, rely on domain expertise, and are defined over static data instances. In more recent work, CurrentClean is a probabilistic framework for identifying and repairing stale cells by modeling spatio-temporal update patterns in the data [75]. Similar to our work, CurrentClean learns from a history of past updates to compute the probability that a cell value is current. However, our work considers replicated, multi-node settings where versioning of a cell value, and

node unavailability can occur. We provide confidence bounded currency estimation in these settings. Sanjay et al., propose SVC [89] to compute fresh answers from stale materialized views (MVs), as changes to the base relation occur and are not propagated to the MVs, leading to an incorrect response to a query. Firstly, for a given ratio m, SVC samples the stale MV. Secondly, SVC uses a provenance-based technique to track the primary key of the changed tuples in the base relation, and aggregate changed tuples with the stale MV sample to produce an up-to-date MV sample. After that, SVC estimates query results over the cleaned sample. In comparison to our work, SVC shares a similar currency semantics to CurrentClean, where staleness is defined as the last update of a cell in a database (or MV) that does not reflect actual changes in the real world. However, in comparison to replica currency estimation, the definition of staleness differs as we consider a replica to be stale when there exists another replica that has a more current value. To estimate the results over the up-to-date sample, SVC presents the bounds and guarantees on different classes of aggregate queries based on the theorem of confidence intervals. However, we bound the confidence of estimation based on the generalization error of a machine learning model. Both SVC and replica currency estimation impose a bound in their techniques. For different classes of aggregate queries, SVC shows the difference between estimated query result and the ground truth is bounded. However, in replica currency estimation, we apply the generalization bound from linear regression models to derive confidence guarantees on our currency estimations.

3.6.2 Data Freshness

Existing work has studied protocols to satisfy user-given freshness guarantees (in the form of absolute time) using freshness locks [3], economic models that assign higher cost to recent updates [27], and controlling the read replica set to provide query answers [101]. A primary requirement and benefit of our work is to incur minimal overhead, making locking-based solutions and re-calculation of read replica sets infeasible.

Probabilistically Bounded Staleness (PBS) [8] detects staleness for query results across a set of replicas. Despite this similar goal, there are several notable differences. First, PBS considers read requests where the last write was committed at least t time units ago, while our study does not have such constraints. Intuitively, PBS may report false positive cases where data is reported as current, but newer, uncommitted data exists since the last commit. We consider such cases in our work. Second, the definition and semantics of staleness are different. When PBS processes the read request (after t time units since the last write), a replica is said to be stale if a read arrives before a write. In contrast, we say a replica is stale if there exists another replica with a more recent (the latest) write.

3.6.3 Data Replication

Past work has proposed techniques to guarantee and improve data freshness under lazy-master replication settings [68, 80]. In eager replication, all replica copies are

simultaneously updated using strategies such as quorum consensus [13] or two-phase-commit leading to mutual consistency and freshness [80]. In contrast, in lazy replication, deferred updates lead to varying data freshness levels across the replicas for unseen update transactions. Adaptive update policies take an on-demand approach based on access frequency, current transaction miss ratio, and system utilization [103]. Our work shares the same goal towards achieving data freshness (currency), but we do not focus on the update propagation strategy. In such systems, data freshness often depends on system conditions such as replica workload, frequency of updates, and network latency. However, our method provides confidence bounded estimates that are adaptive to these system conditions, and can be used together with existing solutions to reduce latency. By exchanging model parameters between nodes, we have shown lower overhead costs than message and update passing between nodes (about only 0.76%-1.17% of the original query processing and replica synchronization time costs).

3.7 Conclusions

In this chapter, we study how machine learning techniques advance replica currency estimation in distributed databases, and enable a novel DYNAMIC consistency level. Remarkably, the confidence of replica currency estimation is theoretically bounded (in Propositions 4 and 5). By referring to the guaranteed confidence of a replica's currency, the system dynamically decides whether to wait for other replicas to respond.

We integrate our techniques in the open-source, distributed database Apache Cassandra [5], and conduct an extensive evaluation under various query loads and cluster configurations. The replica currency estimation has high confidence (at least 0.99 confidence levels) without introducing much overhead (incurring only about 0.76%-1.17% of the original query processing and replica synchronization time costs). The proposed DYNAMIC consistency level is effective and efficient compared to existing consistency levels (ONE, QUORUM, ALL) in Cassandra that retrieve a fixed number of replicas. We believe that the replica currency estimation and the DYNAMIC consistency level are extensible to other (NoSQL) databases, such as HBase [6], and we intend to implement such features as future work.

Chapter 4

OFDClean: Contextual Data Cleaning with Ontology Functional Dependencies

In the last two chapters, we focused on estimating data currency. We now turn our focus to improving data consistency, where consistency is measured by the alignment between a dataset and a set of data dependencies (also generally referred to as data quality rules).

Table 4.1: Sample clinical trials.

id	CC	CTRY	SYMP	TEST	DIAG	MED
t_1	US	USA	joint pain	CT	osteoarthritis	ibuprofen
t_2	IN	India	joint pain	CT	osteoarthritis	NSAID
t_3	CA	Canada	joint pain	CT	osteoarthritis	naproxen
t_4	IN	Bharat	nausea	EEG	migraine	analgesic
t_5	US	America	nausea	EEG	migraine	tylenol
t_6	US	USA	nausea	EEG	migraine	acetaminophen
t_7	IN	India	chest pain	X-ray	hypertension	morphine
t_8	US	USA	headache	CT	hypertension	cartia
t_9	US	USA	headache	MRI	hypertension	tiazac (ASA)
t_{10}	US	America	headache	MRI	hypertension	tiazac
t_{11}	US	USA	headache	CT	hypertension	tiazac (adizem)

4.1 Introduction

In constraint-based data cleaning, dependencies are used to specify data quality requirements. Data that are inconsistent with respect to the dependencies are identified as erroneous, and updates to the data are generated to re-align the data with the dependencies. Existing approaches use Functional Dependencies (FDs) [17, 82], Inclusion Dependencies [17], Conditional Functional Dependencies [29], Denial Constraints [24], Order Dependencies [93], and Matching Dependencies [43] to define the attribute relationships that the data must satisfy. However, these approaches are limited to identifying attribute relationships based on syntactic equivalence (or syntactic similarity for Metric FDs [67, 82]), and unable to convey semantic equivalence, which is often necessary in data cleaning.

Example 8: Table 4.1 shows a sample of clinical trial records containing patient country codes (CC), country (CTRY), symptoms (SYMP), diagnosis (DIAG), and

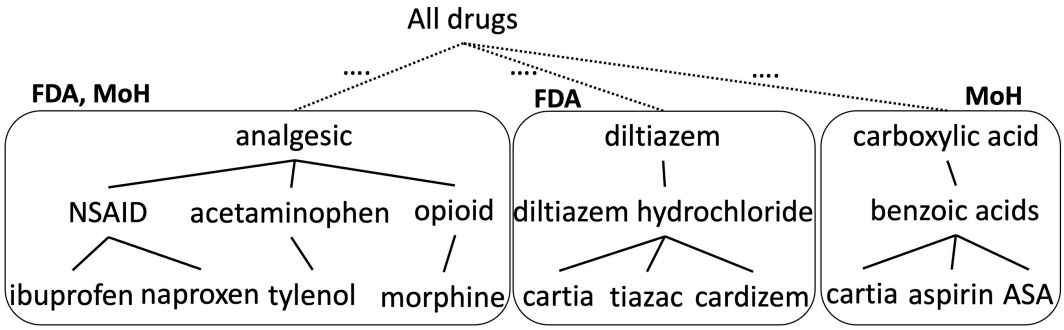


Figure 4.1: Medical drug ontology.

the prescribed medication (MED). Consider two FDs: F_1: [CC] Ñ [CTRY] and F_2: [SYMP, DIAG] Ñ [MED]. The tuples (t_1, t_5, t_6, t_8 t_{11}) do not satisfy F_1 as *America* and *USA* are not syntactically equivalent (the same is true for (t_2, t_4, t_7)). However, *USA* is synonymous with *America*, and (t_1, t_5, t_6, t_8 t_{11}) all refer to the same country. Similarly, *Bharat* in t_4 is synonymous with *India* as it is the country's original Sanskrit name. For F_2, (t_1 t_3), (t_4 t_6) and (t_8 t_{11}) are violations as the consequent values all refer to different medications. However, as shown in Figure 4.1, with domain knowledge from a medical ontology [86], we see that the values participate in an inheritance relationship. Both *ibuprofen* and *naproxen* are non-steroidal anti-inflammatory drugs (*NSAID*), *tylenol* is an *acetaminophen* drug, which in turn is an *analgesic*, and both *cartia* and *tiazac* are *diltiazem hydrochloride*. I

The above example demonstrates that real data contain domain-specific relationships beyond syntactic equivalence or similarity. It also highlights two common relationships that occur between two values u and v: (1) u and v are *synonyms*; and (2) u is-a v denoting *inheritance*. These relationships are often defined within domain-specific ontologies. Unfortunately, traditional FDs and their extensions are unable to

capture these relationships, and existing data cleaning approaches flag tuples containing synonymous and inheritance values as errors. This leads to an increased number of reported "errors", and a larger search space of data repairs to consider.

To address these shortcomings, our earlier work proposed a novel class of dependencies called *Ontology Functional Dependencies* (OFDs) that capture synonyms and is-a relationships defined in an ontology [10]. In this chapter, we focus on synonyms, which are commonly used in practice, and we study cleaning a relation, and an ontology with respect to (w.r.t.) a set of synonym OFDs. What makes OFDs interesting is the notion of *senses*, which determine how a dependency should be interpreted for a given ontology; e.g., *jaguar* can be interpreted as an *animal* or as a *vehicle*, country codes vary according to multiple standards (interpretations) such as the International Standards Organization (ISO) vs. United Nations (UN). To make OFDs useful in practice, where data semantics are often poorly documented and change, we propose an algorithm to discover OFDs.

OFDs serve as contextual data quality rules that enforce the semantics modeled in an ontology. However, application requirements change, data evolve, and as new knowledge is generated, inconsistencies arise between the data, OFDs, and ontologies. For example, the US Food and Drug Administration (FDA) has a monthly approval cycle to certify new drugs. If downstream data applications do not update their data and ontologies, this leads to stale and missing values in the ontology, and inconsistencies w.r.t. the data and the OFDs [86]. Similarly, changes to the data may be required to re-align the data to the OFDs and the ontology. Consider the following example.

Example 9: Consider Table 4.1 with updated values (shown in blue) in t_9[MED] and t_{11}[MED] to reflect changes in a patient's prescribed medication. Tuples (t_8 t_{11}) are now inconsistent w.r.t. the previously defined F_2. If we augment F_2 with additional semantics, provided by a medication ontology as shown in Figure 4.1, tuple t_{11} continues to be problematic since *adizem* is not defined in the ontology, and is not equivalent to *cartia* nor *tiazac*. The updated value in t_9[MED] to *ASA* leads to an inconsistency since *ASA* is not semantically equivalent to *cartia* nor *tiazac*. We must find an interpretation of the ontology, called a *sense* (denoted in bold in Figure 4.1), where the values {*ASA*, *cartia*, *tiazac*, *adizem*} are all equivalent. Unfortunately, there is no sense in which all these values are semantically equivalent. To resolve these violations, we can: (1) repair the ontology by adding the value *adizem* and *ASA* under the FDA sense; or (2) update tuples (t_8 t_{11}) to either *cartia* or *ASA* under the MoH sense. In both cases, there now exists a sense where all MED values in (t_8 t_{11}) are equivalent. ∎

The above example demonstrates that there are multiple options to resolve violations, namely, modifying the ontology or the data. We study the repair problem that re-aligns the data, dependencies (OFDs), and an ontology via a minimal number of repairs to the data and to the ontology. State-of-the-art data cleaning solutions have taken a holistic approach to combine attribute relationships with external knowledge bases and statistical methods to determine the most likely repair [82], and probabilistic models that learn from clean distributions in the data to minimally repair dirty values [99]. However, none of these techniques consider repairs to ontologies. We adopt a similar philosophy to prior work that consider repairs to the data and/or

constraints [23, 51]. We make the following contributions:

1. We present OFDClean, an algorithm that computes a minimal number of modifications to an ontology and to the data to satisfy a given set of OFDs. To consider the possible interpretations of the data, OFDClean selects the best interpretation (sense) for an equivalence class of tuples, such that the selected sense minimizes the number of required modifications.

2. We evaluate the performance and effectiveness of OFDClean using real datasets over varying parameter values. We show the effectiveness of our sense selection algorithm. Compared to HoloClean[83], a holistic data cleaning framework, we achieve increased precision and recall over HoloClean by 7.4% and 4.4%, respectively.

We give preliminary definitions in Section 4.2. We introduce data and ontology repairs, and the OFDClean framework in Section 4.3. In Section 4.4, we describe our sense selection algorithm, and then present our ontology repair algorithm in Section 4.5. We present experimental results in Section 4.6, related work in Section 4.7, and conclude in Section 4.8.

4.2 Preliminaries and Definitions

Functional Dependencies. A functional dependency (FD) F over a relation R is represented as X Ñ A, where X is a set of attributes and A is a single attribute in R.

An instance I of R satisfies F if for every pair of tuples t_1, t_2 P I, if $t_1[X] = t_2[X]$, then $t_1[A] = t_2[A]$. A partition of X, Π_X, is the set of equivalence classes containing tuples with equal values in X. Let x_i be an equivalence class with a representative i that is equal to the smallest tuple id in the class, and $|x_i|$ be the size of the equivalence class (in some definitions, we drop the subscript and refer to an equivalence class simply as x). For example, in Table 4.1, $\Pi_{CC} = \{\{t_1, t_5, t_6\}\{t_2, t_4, t_7\}\{t_3\}\}$.

Sense. An ontology S is an explicit specification of a domain that includes concepts, entities, properties, and relationships among them. In this work, we consider tree-shaped ontologies for simplicity. These constructs are often defined and applicable only under a given interpretation, called a *sense*. The meaning of these constructs for a given S can be modeled according to different senses leading to different ontological interpretations. As mentioned previously, the value *jaguar* can be interpreted under two senses: (1) as an animal, and (2) as a vehicle. As an animal, *jaguar* is synonymous with *panthera onca*, but not with the value *jaguar land rover automotive*.

We define classes E for the interpretations or senses defined in S. Let *synonyms*pEq be the set of all synonyms for a given class E. For instance, *synonyms*p$E1$q = {*car, auto, vehicle*}, *synonyms*p$E2$q = {*jaguar, jaguar land rover*} and *synonyms*p$E3$q = {*jaguar, panthera onca*}. Let *names*pCq be the set of all classes, i.e., interpretations or senses, for a given value C. For example, *names*p*jaguar*q = {$E1$, $E2$, $E3$} as jaguar can be an animal or a vehicle. Let *descendants*pEq be a set of all representations for the class E or any of its descendants, i.e., *descendants*pEq = {s | s P *synonyms*pEq or s P *synonyms*pE_iq, where E_i *is-a* $E_{i\ 1}$, ..., E_1 *is-a* E}, e.g., *descendants*p$E3$q = {*jaguar, peruvian jaguar,mexican jaguar*}.

Ontology Functional Dependencies We define OFDs w.r.t. a given ontology S. We consider the synonym relationship on the right-hand-side of a dependency leading to synonym OFDs.

Definition 2.1: A relation instance I satisfies a *synonym OFD* X Ñ$_{syn}$ A, if for each equivalence class x P Π_XpIq, there exists a sense (interpretation) under which all the A-values of tuples in x are synonyms. That is, X Ñ$_{syn}$ A holds, if for each equivalence class x,

£ H.

$names$pa$q$,aP$t$tr$A$s|tPxu

I

Example 10: Consider the OFD [CC] Ñ$_{syn}$ [CTRY] from Table 4.1. We have Π_{CC} = $\{\{t_1, t_5, t_6\}\{t_2, t_4, t_7\}\{t_3\}\}$. The first equivalence class, $\{t_1, t_5, t_6\}$, representing the value *US*, corresponds to three distinct values of CTRY. According to a geographical ontology, names(*United States*) X names(*America*) X names(*USA*) = *United States of America*. Similarly, the second class $\{t_2, t_4, t_7\}$ gives names(*India*) X names(*Bharat*) = *India*. The last equivalence class $\{t_3\}$ contains a single tuple, so there is no conflict. Since all references to CTRY in each equivalence class resolve to some common interpretation, the synonym OFD holds. I

Synonym OFDs subsume traditional FDs, where all values are assumed to have a single literal interpretation (for all classes E, |*synonyms*pEq| = 1). Similar to traditional FDs, we say that an instance I satisfies a set of OFDs Σ, denoted I |ù Σ *if and only if* I satisfies each OFD ϕ P Σ. Without loss of generality, we assume that

the consequent (right side) of each ϕ P Σ consists of a single attribute A (this follows from the axioms that can be found in the full version [111]). When we refer to an OFD ϕ: X Ñ Y, we say I |ù ϕ iff I |ù pX Ñ Aq for all A P Y. That is, for each attribute A P Y, all the A-values of tuples in x share a common sense (interpretation).

Ontological relationships can be semantically meaningful on both sides of a dependency to capture a greater number of true positive errors. However, due to the large search space of ontological relationships on the left-hand side, since we must also consider all tuples outside of the partition group when checking for violations, we focus on the right-hand-side. This is similar to other seminal work [67, 82] that considers small variations only on the right-hand-side via metric FDs.

4.3 Data Cleaning with OFDs

Over time, misalignment may arise between a data instance I, an ontology S, and a set of OFDs Σ. In this section, we study the problem of how to compute repairs to re-align I, S with Σ. Data naturally evolve due to updates and changes in domain semantics. These changes in semantics also lead to ontology incompleteness, as new concepts and ontological relationships are introduced. For example, new uses of medical drugs lead to new prescriptions to treat new and different illnesses. While OFDs may also evolve, we argue that repairs to the data I and ontology S are more common in practice, and hence, our current focus. When stale OFDs do occur, FastOFD can be used to discover the latest set of OFDs that hold over the data. For approximate OFDs defined over a dirty instance I, violating values in I can be

repaired, thereby transforming approximate OFDs to OFDs that are satisfied over all tuples in I. Formally, we modify I and S to produce I^1 and S^1, such that a repaired version of I, I^1 |ù Σ, w.r.t. a modified version S^1 of S.

4.3.1 Scope of Repairs

OFDs interact when two or more dependencies share a common set of attributes, which may invalidate already performed repairs. For simplicity, we assume that there does not exist an attribute that occurs on the left side of one OFD and the right side of another. This simplifies the interaction among repairs to only consequent attributes without having to consider changes to antecedent (left-side) attributes of the dependency, i.e., the equivalence classes for an OFD remain fixed. In many domains, attributes serve as either an independent or dependent role. For example, in health care, demographic and prognosis characteristics are typically independent and dependent attributes, respectively. However, our techniques can handle OFDs that share the same consequent attribute.

Data Repair. Consider a synonym OFD ϕ: X Ñ$_{syn}$ A, and an equivalence class x P Π_XpIq where there exists at least two tuples t_1, t_2 P x, such that t_1rAs and t_2rAs are not synonyms under some sense λ. We consider repairs to values in the consequent attributes where the domain of repair values is values in S. Let $\mathcal{P}$pIq denote the set of all possible data repairs of I. For a repair I^1 of I, we use $dist$pI, I^1q to denote the difference between I and I^1, which is measured by the number of cells whose values in I are different from the values of the corresponding cells in I^1.

Ontology Repair. A repair to an ontology S is the insertion of new value(s) to a node in S w.r.t. a sense λ. Let $\mathcal{P}$pSq denote the set of possible ontology repairs to S. For a repair S^1 of S, we define distance $dist$pS, S^1q as the number of new values (*concepts*) in S^1 that are not in S.

Minimal Repair. The universe of repairs, U, represents all possible pairs (S^1, I^1) such that S^1 P $\mathcal{P}$pSq and I^1 P $\mathcal{P}$pIq and I^1 |ù Σ w.r.t. S^1. We want to find minimal repairs in U that do not make unnecessary changes to S or I in a Pareto-optimal sense. A repair is considered Pareto-optimal if there does not exist any other repair that dominates it. Specifically, pS^1_i, I^1_iq dominates pS^1_j, I^1_jq, denoted pS^1_i, I^1_iq pS^1_j, I^1_jq, if t@i, j|p$dist$pS, S^1_iq ¤ $dist$pS, S^1_jqq, p$dist$pI, I^1_iq ¤ $dist$pI, I^1_jqqu, and tDk|p$dist$pS, S^1_iq $dist$pS, S^1_kqq _ p$dist$pI, I^1_iq $dist$pI, I^1_kqqu.

The concept of dominance can be defined generally for two vectors as follows.

Definition 3.1: We define vectors $\mathcal{X}$ p$x_1, ..., x_m$q and $\mathcal{Y}$ p$y_1, ..., y_m$q, and say $\mathcal{X}$ dominates $\mathcal{X}$, denoted by $\mathcal{X}$ $\mathcal{Y}$, iff for i P $1, ..., m, x_i$ ¤ y_i, and at least one element x_j in $\mathcal{X}$ is strictly less than the corresponding element y_j in $\mathcal{Y}$. |

Definition 3.2: (*Minimal Repair*). We are given an instance I, OFDs Σ, and an ontology S, where I |ù Σ *w.r.t.* S. A repair (S^1, I^1) P U is minimal if EpS^2, I^2q P U such that $dist$pS, S^2q $dist$pS, S^1q and $dist$pI, I^2q $dist$pI, I^1q. |

For example, suppose U contains three repairs: the first one makes two changes each to S and I, the second one makes two changes to S and three changes to I, and the third one makes one change to S and five changes to I. Here, the first and the

third repair are minimal.

Problem: *For a data instance I, an ontology S, a set of OFDs Σ, where I |ù Σ, compute a repaired instance I^1, and ontology S^1, such that I^1 |ù Σ w.r.t. S^1, and dist$\mathsf{p}I, I^1\mathsf{q}$ and dist$\mathsf{p}S, S^1\mathsf{q}$ are minimal.*

4.3.2 OFDClean Overview

OFDs enrich the semantic expressiveness of FDs via ontological senses, which allow for multiple interpretations of the data. In traditional FDs, senses do not exist, and there is only one interpretation. For a set of OFDs Σ, the problem of finding a minimal repair to S and to I requires evaluating candidate senses for each equivalence class w.r.t. each ϕ P Σ, and assigning a sense to each class.

Existing constraint-based repair algorithms are insufficient as they do not consider senses [23, 48]. We therefore propose OFDClean, a framework that includes ontology and data repair with sense assignment to minimize the number of changes to I, and S. Intuitively, we consider ontology repairs of size k, where k ranges from one to the total number of ontology repair values. For each candidate ontology repair of size k, we select the ontology repair S^1 such that dist$\mathsf{p}S, S^1\mathsf{q}$ is minimal. For each value k, and the selected S^1, we compute a repair I^1 of I such that the number of updates is minimal, and less than a threshold τ, i.e., dist$\mathsf{p}I, I^1\mathsf{q}$ ¤ τ to achieve consistency with Σ. We call such repairs *τ-constrained repairs*. To generate a Pareto-optimal set of repairs, we select $\mathsf{p}S^1, I^1\mathsf{q}$ that are minimal for each k value. A naive approach first evaluates all candidate senses (denoted by m) against all x P Π_X, and the complexity is $\mathsf{p}|\Pi_X|\mathsf{q}^m$,

where m is the number of senses. Secondly, for each k-combination of ontology repairs, it computes $\mathsf{p}S^1, I^1\mathsf{q}$ that are minimal, and the complexity is $2^{|k|}$. Hence, the complexity of generating a Pareto-optimal set of repairs is $\mathsf{p}|\Pi_X|\mathsf{q}^m \quad 2^{|k|}$, which is not feasible in practice. Thus, we propose a greedy-based framework, OFDClean, that reduces the complexity from exponential to polynomial.

OFDClean takes a greedy, two-step approach. First, we compute an initial sense assignment for each equivalence class that minimizes the number of repairs to the ontology and to the data (locally) w.r.t. each OFD and each equivalence class. Secondly, we go beyond a single OFD to consider pairwise interactions among ϕ P Σ where attribute overlap occurs, such that repairing values w.r.t. ϕ_1 may increase or decrease the number of errors in ϕ_2. We model the overlapping tuples between ϕ_1 and ϕ_2 as two distributions w.r.t. their assigned senses. We refine the sense assignment such that the number of necessary updates to transform one distribution to the other is minimized. Lastly, with a computed sense assignment, our problem reduces to finding a minimal repair of S and to I such that it satisfies Σ. Figure 4.2 provides an overview of OFDClean. Given I, S and Σ, OFDClean identifies inconsistencies in I w.r.t. Σ and S, and returns a set of minimal repairs (I^1, S^1) such that I^1 |ù Σ w.r.t. S^1. OFDClean execution proceeds along two main modules.

Sense Assignment. Let $\phi : X$ Ñ$_{syn}$ A be a synonym OFD, where for each equivalence class x P Π_X, x is interpreted w.r.t. an assigned sense λ_x. Given I, S, and Σ, for each ϕ P Σ, we compute λ_x for each x P Π_X such that $\mathsf{p}I^1, S^1\mathsf{q}$ is minimal and I^1 |ù ϕ. We denote the set of all senses across all x for ϕ as $\Lambda\mathsf{p}\phi\mathsf{q} \quad \mathsf{Y}_{x\mathsf{P}\Pi\mathsf{p}X\mathsf{q}}\lambda_x$. Intuitively, we compute λ_x for every x P Π_X to obtain an interpretation that can achieve a minimal

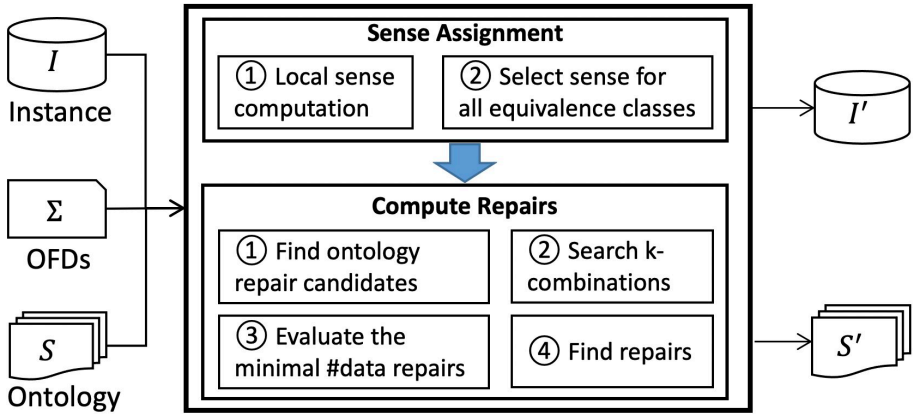


Figure 4.2: OFDClean framework.

repair. We take a local greedy approach to consider only ϕ when computing $\Lambda\mathfrak{p}\phi\mathfrak{q}$. We then model the interactions among the OFDs in Σ, via their overlapping equivalence classes, and refine the initial sense assignment. We quantify interactions and conflicts by comparing the difference between the distributions represented via a pair of equivalence classes. The final assignment of senses for all ϕ P Σ is denoted as $\Lambda\mathfrak{p}\Sigma\mathfrak{q}$ Y_{ϕ} $\Lambda\mathfrak{p}\phi\mathfrak{q}$. We present our sense assignment algorithm in Section 4.4.

Compute Repairs. Given $\Lambda\mathfrak{p}\Sigma\mathfrak{q}$, Compute Repairs computes the set of minimal repairs by first generating the set of ontology repair candidates that appear in an equivalence class x, but not in S. It then considers combinations of these candidates. We optimize lattice traversal using a beam-search strategy that selects the top-b candidates to expand at each level of the lattice [70]. We consider data updates to the consequent attribute A from the domain of A and synonyms of A in S under the selected sense. We then select candidate repairs that make at most τ updates. We present details of how ontology and data repairs are computed in Section 4.5.

4.4 Sense Assignment

Given senses for an ontology S, we compute λ_x for each x and ϕ P Σ such that each repair $\mathsf{p}I^1, S^1\mathsf{q}$ is minimal and τ-constrained. A naive approach evaluates all candidate senses against all x P Π_X, leading to an exponential number of solutions, which is not feasible in practice.

In this section, we introduce a two-step greedy approach. First, we build an initial solution by considering all senses against each equivalence class x, and selecting the sense that minimizes the number of repairs for each x P Π_X and for each $\phi : X$ $\tilde{\mathsf{N}}_{syn}$ A, leading to a local minimum. This evaluation greedily selects the sense that leads to the fewest repairs for a given x, without considering interactions between the equivalence classes and their senses. In the second step, we consider the interactions among pairs of equivalence classes x, x^1, where x P Π_X, x^1 P Π_{X^1} for ϕ, ϕ^1 P Σ, respectively, by performing a local search to improve the initial solution w.r.t. minimizing the number of repairs. Our greedy search selects pairs of equivalence classes where the modeled data distributions share the greatest distance, indicating a larger amount of conflict. We model the interactions among all ϕ, ϕ^1 P Σ that share a common consequent attribute by constructing a *dependency graph*, where nodes represent equivalence classes and edges represent a shared consequent attribute. We traverse this dependency graph to refine the initial solution. We evaluate the tradeoff between updating the initial sense assignment and a new assignment to x by estimating the number of necessary data and ontology repairs to achieve alignment between x and x^1. We visit nodes x^1 within a neighbourhood of the current node x as defined via a distance function. We use the *Earth Mover's Distance (EMD)* to quantify the amount of work needed to transform

the data distributions modeled in x to the distribution represented in x^1 [88]. We visit nodes x^1 where the EMD values between x and x^1 are largest, indicating regions with the greatest amount of conflict, while avoiding exhaustive enumeration of all candidate assignments. We first describe our greedy algorithm to generate an initial solution, followed by local neighbourhood refinement.

4.4.1 Computing an Initial Assignment

For $\phi : X \tilde{\mathsf{N}}_{syn} A$, $x \mathsf{P} \Pi_X$, we compute λ_x, an initial sense assignment for each x. To compute a minimal repair, we seek senses λ_x containing as many values as possible (maximal overlap) with values in x. Let t_u, v_u represent the number of distinct tuples and values, respectively, that are not covered by λ_x. Let $\overline{r}_{\lambda_x}$ $\mathsf{p}t_u, v_u\mathsf{q}$ be a vector representing the number of necessary tuples and values that must be updated in x under sense λ to satisfy ϕ. We seek a sense λ^1 such that $\overline{r}_{\lambda^1_x}$ $\overline{r}_{\lambda_x}, @\lambda_x$.

The enumeration of all candidate senses to x is practically inefficient. We take a greedy approach that finds a λ_x with maximal overlap with values in x. We first construct an index of all senses containing each value $v \mathsf{P} x$, denoted by $sset\mathsf{p}v\mathsf{q}$. For example, Figure 4.3(c) shows $sset\mathsf{p}c_3\mathsf{q}$ $\mathsf{t}\lambda_1, \lambda_4\mathsf{u}$. To maximize the coverage of values, we naturally seek values with maximal frequency. However, given that values in x may contain errors, we seek an ordering of the values in x that is robust to outliers. The *Median Absolute Deviation (MAD)* measure quantifies the variability of a sample, being more resilient to outliers than the standard deviation [87]. Let $f\mathsf{p}v\mathsf{q}$ denote the frequency of value v. For a univariate dataset $\mathsf{t}f\mathsf{p}v_1\mathsf{q}, f\mathsf{p}v_2\mathsf{q}, \ldots f\mathsf{p}v_n\mathsf{qu}$, MAD is

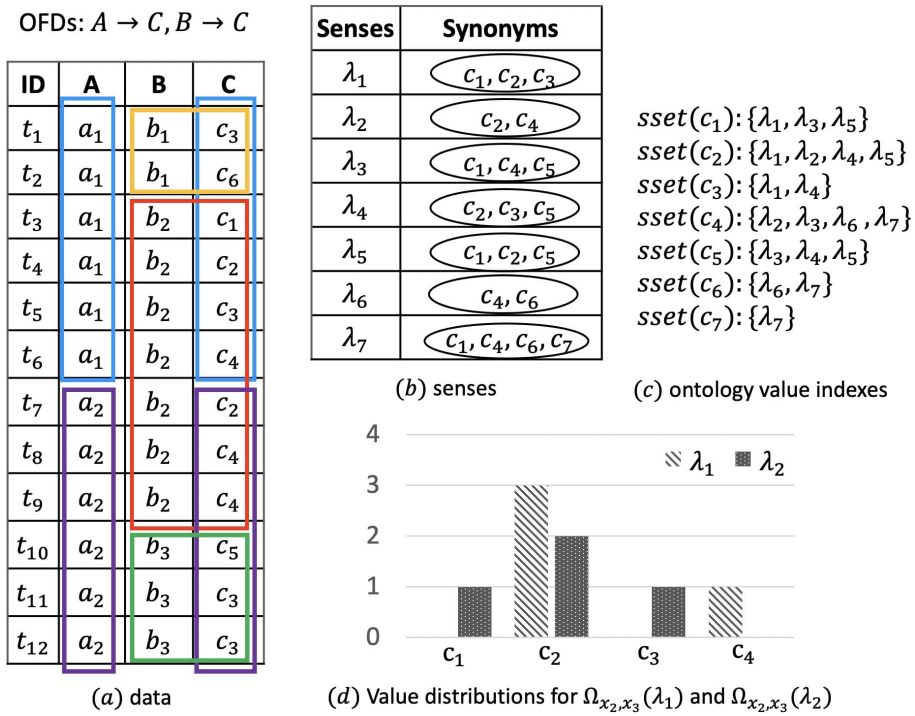


Figure 4.3: Sense assignment.

defined as the median of the absolute deviations from the median, i.e., MAD(x) = median$\mathsf{p}|f\mathsf{p}v_i\mathsf{q} \quad f\mathsf{pp}\bar{v}\mathsf{qq}|\mathsf{q}$, where $f\mathsf{pp}\bar{v}\mathsf{qq}$ is the median of all values in x. We sort the values in x according to their decreasing MAD scores. We iteratively search for a sense λ_x containing as many values as possible from x with the highest MAD scores. We use k^1 as a (decreasing) counter to find such a sense covering k^1 values in x, where k^1 is initialized to n, the number of unique values in x. For each value v_i, we compute the intersection of their respective $sset\mathsf{p}v_i\mathsf{q}$. If there exists a non-empty intersection of senses, the algorithm ends, and returns these senses as an assignment for x, denoted as $potential_set\mathsf{p}x\mathsf{q}$. We select the sense λ_x covering the largest number of tuples in x. Algorithm 2 provides the details.

Algorithm 2: Initial_Assignment(x, S, Λ)

```
let sset(v) ← ∅
for each λ ∈ Λ do
    for each value v ∈ λ do
        sset(v) ← sset(v)    λ
    end for
end for
k^1 ← MAD(x)
potential_set(x) ← ∅
while potential_set(x) is ∅ do
    for each top-k^1
    {f(v_1), ..., f(v_n)} ⊆ MAD(x) do
        Λ^1 ← sset(v_1)    ...    sset(v_n)
        if Λ^1 ≠ ∅ then
            potential_set(x) ←
            potential_set(x)    Λ^1
        end if
    end for
    k^1 ← k^1 − 1
end while
for each sense λ ∈ potential_set(x) do
    cover(λ) ← the tuple coverage of λ over x
end for
λ_x ← λ ∈ cover(λ) with maximal tuple
coverage
return λ_x
```

Algorithm 3: Local_Refinement(G, x, Λ)

```
u_1 ← BFS(G)
for each vertex u_2 connected to u_1 do
    if w(u_1, u_2) > θ then
        compute min{cost} repair
        if sense reassignment for u_2 to λ^1
        then
            if w^1(u_1, u_2) < w(u_1, u_2) then
                Λ ← Λ(I)\{u_2, λ}
                Λ ← Λ(I)    {u_2, λ^1}
            end if
        end if
    end if
end for
continue until all all u_1 visited
return Λ
```

Example 11: Figure 4.3(a) shows a data instance I for two synonym OFDs $\phi_1 : A$ Ñ C and $\phi_2 : B$ Ñ C. Figure 4.3(b) shows seven senses and the corresponding synonym values for each sense. Let λ_{x_3} represent equivalence class x_3 Π_B $_{b_2}$. Figure 4.3(c) shows the index of senses for each value for all x. We compute the MAD for values in x_3, i.e., c_1 c_4, and determine that the ranked ordering is tc_4, c_2, c_1, c_3u. We intersect the senses in $sset$pc_2q and $sset$pc_4q, resulting in λ_2 as the initial sense for x_3. l

4.4.2 Local Refinement

We now discuss how we model interactions between x and x^1, and address conflicting sense assignments that can lead to an increased number of repairs.

Modeling Interactions

Let $\Omega_{x,x^1}\mathsf{p}\lambda\mathsf{q}$ $\mathsf{t}t|t$ P $\mathsf{p}x$ X $x^1\mathsf{qu}$ represent the set of overlapping tuples between two equivalence classes x, x^1 from ϕ, ϕ^1, respectively, that share the same consequent attribute, interpreted w.r.t. sense λ. To quantify conflict among ϕ, ϕ^1, we need to interpret tuple values in the intersection of x and x^1 under the assigned sense. That is, conflicts may arise when there is a difference in the interpretation of the tuple values. Let x_λ represent the set of values in x when interpreted under sense λ. Given synonym relations, assume that for each sense λ, there exists a canonical value representing all equivalent values in λ. Let $D\mathsf{p}x_\lambda\mathsf{q}$ represent the distribution of values in x_λ, where values in x that are in λ are represented by the canonical value. If x and x^1 are assigned the same sense λ, then tuples in their intersection, $(\Omega_{x,x^1}\mathsf{p}\lambda\mathsf{q})$, share the same distribution, $D\mathsf{p}\Omega_{x,x^1}\mathsf{p}\lambda\mathsf{qq}$. When it is clear from the context, we use $\Omega\mathsf{p}\lambda\mathsf{q}$to denote overlapping tuples in x and x^1.

Example 12: Let x_2 $\Pi_{A\ a_1}$ and x_3 $\Pi_{B\ b_2}$, highlighted in blue and red, respectively, in Figure 4.3(a). If we assign sense λ_1 to both classes, then the tuples in $\Omega_{x_2,x_3}\mathsf{p}\lambda_1\mathsf{q}$ $\mathsf{t}t_3, t_4, t_5, t_6\mathsf{u}$ share the same distribution, as shown via light grey bars in Figure 4.3(d), assuming that c_2 is the canonical value for λ_1. I

The above example demonstrates that when two distributions, $D(\Omega p\lambda q)$ and $Dp\Omega p\lambda^1 qq$, are equivalent for $\lambda \quad \lambda^1$, the *outliers*, i.e., values not captured by λ, translate into necessary repairs to achieve consistency w.r.t. tuples in Ω. In the above example, c_4 is the outlier, and we can resolve the inconsistency by updating c_4 to c_2 or by adding c_4 to λ_1. In contrast, when $\lambda \quad \lambda^1$, tuples in $\Omega p\lambda q$and $\Omega p\lambda^1 q$ will share different outliers. For example, consider assigning λ_1 to x_2, and λ_2 to x_3. The distributions for $\Omega_{x_2,x_3} p\lambda_1 q$ and $\Omega_{x_2,x_3} p\lambda_2 q$ are shown via the light grey bars and dark grey bars, respectively, in Figure 4.3(d). Assuming c_2 is the canonical value for each sense, the outliers are c_1 and c_3 for λ_2, and c_4 for λ_1. Let $\rho_{\Omega,\lambda}$ represent the set of outliers w.r.t. sense λ, i.e., the unique values in $\Omega p\lambda q$ not covered by λ, $\rho_{\Omega,\lambda}= tv|v \ P \ \Omega_{x,x^1} p\lambda q, v \ R \ \lambda u$. We can resolve these outliers by: (1) updating the sense of x from λ to a new λ^1; (2) adding outlier values $\rho_{\Omega,\lambda}$ to S; or (3) performing data repairs to update the values in $\rho_{\Omega,\lambda}$ to values $v \ P \ \lambda$.

For a class x, we seek a new sense assignment $\lambda^2 \quad \lambda$ that includes one or more values in $\rho_{\Omega,\lambda}$. This will cause an increase in the number of data or ontology repairs, as this will require sacrificing at least one value $trAs$ for attribute A that was covered under λ, $t \ P \ tx z\Omega_{x,x^1} p\lambda qu$. This penalty occurs since if there exists a better sense λ^2 that covered all values in $\rho_{\lambda,x}$ then λ^2 would have been assigned to x during the initialization step (Section 4.4.1). Hence, we can have x keep λ or be re-assigned to λ^2. However, we must consider the tradeoff of this re-assignment, by incurring a possible penalty (an increased number of repairs) according to the non-overlapping tuples in x. Alternatively, if x keeps λ, then we must update the outlier values via data and/or ontology repairs.

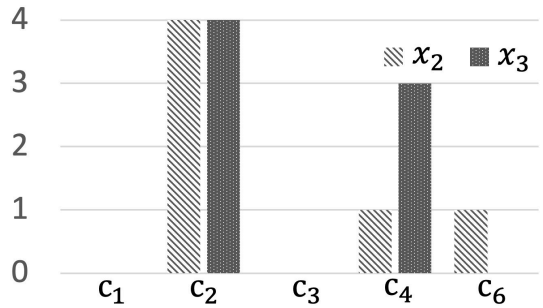


Figure 4.4: $x_{2\lambda_1}$, $x_{3\lambda_1}$ distributions.

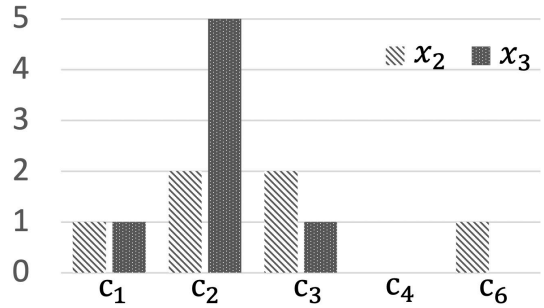


Figure 4.5: $x_{2\lambda_2}$, $x_{3\lambda_2}$ distributions.

Achieving Alignment. To evaluate the tradeoff of a sense re-assignment, we define a cost function cost that sums the number of data and ontology repairs to align $D(\Omega\mathsf{p}\lambda\mathsf{q})$ and $D\mathsf{p}\Omega\mathsf{p}\lambda^1\mathsf{qq}$. For ontology repairs, to re-align $D(\Omega\mathsf{p}\lambda\mathsf{q})$ and $D\mathsf{p}\Omega\mathsf{p}\lambda^1\mathsf{qq}$, we must add each outlier value in $\rho_{\Omega,\lambda}\mathsf{Y}\rho_{\Omega,\lambda^1}$ to ontology S, via senses λ and λ^1, i.e., cost= $|\rho_{\Omega,\lambda}| + |\rho_{\Omega,\lambda^1}|$. Since ontology repairs do not lead to further conflicts, they incur a cost proportional to the number of values added to λ and S. For data repairs, we resolve differences by updating tuples $\mathsf{t}t|t$ P $\Omega\mathsf{p}\lambda\mathsf{q}$, $t\mathsf{r}A\mathsf{s}$ P $\rho_{\Omega,\lambda}\mathsf{Y}\rho_{\Omega,\lambda^1}\mathsf{u}$ to a value from the candidate set $\mathsf{p}\lambda\mathsf{X}\lambda^1\mathsf{q}$, which guarantees that the repaired value is covered by both senses. We set cost $|\mathcal{R}(\Omega\mathsf{p}\lambda\mathsf{q})|$ $|\mathcal{R}\mathsf{p}\Omega\mathsf{p}\lambda^1\mathsf{qq}|$, where $\mathcal{R}(\Omega\mathsf{p}\lambda\mathsf{q}) = \mathsf{t}t|t$ P$\Omega\mathsf{p}\lambda\mathsf{q}$, $t\mathsf{r}A\mathsf{s}$ P $\rho_{\Omega,\lambda}\mathsf{u}$, i.e., the number of tuples containing an outlier value. We select the repair value from the candidate set that minimizes cost. Lastly, we consider sense re-assignment that updates λ to λ^1 for x. For t P x, the delta cost, i.e., the number of data repairs to replace λ with λ^1 is cost $|\mathcal{R}\mathsf{p}x_{\lambda^1}\mathsf{q}|$ $|\mathcal{R}\mathsf{p}x_{\lambda}\mathsf{q}|$, where $\mathcal{R}\mathsf{p}x_{\lambda}\mathsf{q}$ $\mathsf{t}t|t$ P $x, t\mathsf{r}A\mathsf{s}$ P $\rho_{x,\lambda}\mathsf{u}$. For a pair of classes x, x^1, we consider all three repair options, and select the option that locally minimizes cost.

Example 13: Figures 4.4 and 4.5 show the distribution of values in x_2 and x_3 with initial senses λ_1 and λ_2, respectively. Let c_2 be the canonical value for λ_1 and λ_2.

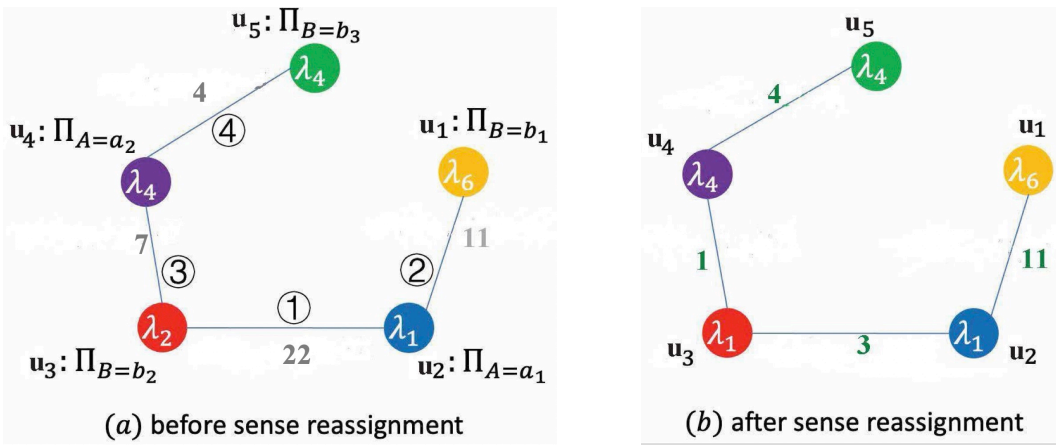


Figure 4.6: Dependency graph.

We have: ρ_{x_2,λ_1} $\mathsf{t}c_4, c_6\mathsf{u}$, ρ_{x_3,λ_1} $\mathsf{t}c_4\mathsf{u}$, and ρ_{x_2,λ_2} $\mathsf{t}c_1, c_3, c_6\mathsf{u}$, ρ_{x_3,λ_2} $\mathsf{t}c_1, c_3\mathsf{u}$. We evaluate ontology and data repair, and two sense assignment options: (i) add c_4 to λ_1, and add c_1 and c_3 to λ_2 to eliminate the conflict, leading to cost= 3; (ii) update three tuples with values $\{c_4, c_1, c_3\}$ to c_2, giving cost= 3; (iii) assign λ_1 to x_3 (from λ_2), leading to $|\mathcal{R}\mathsf{p}x_{3,\lambda_1}\mathsf{q}|$ $|\mathcal{R}\mathsf{p}x_{3,\lambda_2}\mathsf{q}|$ 3 2 1; (iv) assign λ_2 to x_2 where we update $\mathsf{t}c_1, c_3, c_6\mathsf{u}$ to c_2 giving cost= $|\mathcal{R}\mathsf{p}x_{2,\lambda_2\mathsf{q}}|$ $|\mathcal{R}\mathsf{p}x_{2,\lambda_1\mathsf{q}}|$ 4 2 2. Since cost is minimal with option (iii), we re-assign λ_1 to class x_3.

I

Identifying Candidate Classes

In this section, we discuss how to select pairs of classes x, x^1 for refinement. Visiting all pairs is not feasible, as we must evaluate m choose 2 pairs of classes, where m equals the total number of equivalence classes across all ϕ P Σ. For classes $x_\lambda, x^1_{\lambda^1}$, we quantify

the deviation between their respective distributions, by measuring the amount of work needed to transform $D\mathsf{p}\Omega\mathsf{p}\lambda\mathsf{qq}$ to $D\mathsf{p}\Omega\mathsf{p}\lambda^1\mathsf{qq}$ (or vice-versa). We quantify this work using the Earth Mover's Distance measure, denoted EMD [102].

Let P and Q represent distributions $D\mathsf{p}\Omega_{x_1,x_2}\mathsf{p}\lambda_1\mathsf{qq}$, $D\mathsf{p}\Omega_{x_1,x_2}\mathsf{p}\lambda_2\mathsf{qq}$, respectively. We compute $EMD\mathsf{p}P, Q\mathsf{q}$ and check whether $EMD\mathsf{p}P, Q\mathsf{q}$ ¡ θ, for a user-given threshold θ. Intuitively, if P and Q are sufficiently different (according to θ), then classes x_1, x_2 are candidates to consider sense re-assignment to re-align P and Q. Towards finding the sense with minimal repair, if the cost cost of sense re-assignment and the EMD values are lower with the new sense, we proceed with sense re-assignment for x_1, x_2. Otherwise, we keep the initial senses for x_1 and x_2.

Dependency Graph. Let G $\mathsf{p}V, E\mathsf{q}$ denote a *dependency graph*, where each vertex u P V represents an equivalence class x, and an edge e $\mathsf{p}u_1, u_2\mathsf{q}$ P E exists when the corresponding classes $\mathsf{p}x_1$ X $x_2\mathsf{q}$ H. Recall that we define overlap between two classes x_1 and x_2 when their respective OFDs share a common consequent attribute. Hence, we only include nodes (and edges) in the dependency graph for OFDs with a common consequent attribute, thereby avoiding enumeration of all pairs of classes. We use the terms u_1, u_2 to represent classes when it is clear from the context. Let $w\mathsf{p}u_1, u_2\mathsf{q}$ denote the weight of edge e, representing the EMD value between u_1 and u_2. We visit nodes in a breadth-first search manner, and select nodes u_1 with the largest EMD value by summing over all edges containing u_1. This strategy prioritizes classes that require the largest amount of work. For each visited edge, we check whether $w\mathsf{p}u_1, u_2\mathsf{q}$ ¡ θ, and if so, check whether a sense re-assignment will reduce the weight, i.e., reduce the EMD value under the new sense. If so, then the new sense

re-aligns the two class distributions and incurs a minimal cost cost. The algorithm ends when all vertices have been visited. Algorithm 3 provides the details.

Example 14: Figure 4.6(a) shows the dependency graph corresponding to the equivalence classes in Figure 4.3(a), with EMD values as edge weights. Suppose θ 10, and we visit nodes in BFS order of $\mathsf{t}u_2, u_1, u_3, u_4, u_5\mathsf{u}$. Starting at node u_2 (the blue node), we evaluate edge $\mathsf{p}u_2, u_3\mathsf{q}$ with weight 22. From Example 13, we update the sense for u_3 from λ_2 to λ_1 since the new weight $w^1\mathsf{p}u_3, u_2\mathsf{q}$ $w\mathsf{p}u_3, u_2\mathsf{q}$ 3 22. We next consider edge $\mathsf{p}u_2, u_1\mathsf{q}$ with $w\mathsf{p}u_2, u_1\mathsf{q}$ 11, with costs to update u_1 for data repair, ontology repair and sense reassignment as 1, 1 and 0, respectively. However, after updating the sense for u_1 from λ_6 to λ_1, $w^1\mathsf{p}u_2, u_1\mathsf{q}$ does not decrease. Thus, we do not refine the sense for u_1, and keep λ_6 as is. The algorithm then visits node u_3 and evaluates edge (u_3, u_4), where after sense reassignment to λ_1 for u_3, we have $w\mathsf{p}u_3, u_4\mathsf{q}$ 1 θ, and there is no further evaluation needed. Lastly, we visit nodes u_4, u_5, where $w\mathsf{p}u_4, u_5\mathsf{q}$ 4 θ, and the algorithm terminates. Figure 4.6(b) shows the final sense assignments. ∎

4.4.3 Algorithm

Algorithm 4 presents the details of our sense assignment algorithm. We first compute an initial assignment for every equivalence class x. We construct the dependency graph G, and compute the EMD between overlapping classes (u_1, u_2) as edge weights. We visit nodes in decreasing order of their EMD values by summing over all corresponding edges. For example, Figure 4.3(d) shows that we visit u_2 first since the

Algorithm 4: Sense_Assignment(I, Σ, S, θ)

```
Λ ← {∅, ∅}
for each φ ∈ Σ do
    for each x ∈ Π_X(I) do
        for each λ ∈ Λ do
            λ_x ← Initial_Assignment(x, S, Λ)
            Λ ← Λ ∪ {x, λ_x}
        end for
    end for
end for
G = CreateDepGraph(I, Σ, S, Λ)
sort(G, DSC) Sort vertices in DSC order EMD edge weights
while (u_1 = BFS(G) and EMD(u_1) ¿ θ) do
    Λ ← Local_Refinement(G, x, Λ)
end while
return Λ
```

Algorithm 5: Ontology_Repair (I, Σ, S, b)

```
Cand(S) ← ∅
P(S) ← {∅, ..., ∅}
for each φ ∈ Σ do
    for each x ∈ Π_X(I) do
        if t_e ∈ x and t_e[A] ∉ S then
            Cand(S) ← Cand(S) ∪ t_e[A]
        end if
    end for
end for
k ← 1
Generate V_k as clusters of size k, with values v ∈ Cand(S)
L_k ← {v_k | v_k ∈ V_k}
while L_k ≠ ∅ do
    δ(v_k) = compute data repairs for v_k
    L_k ← top-b nodes with smallest δ(v_k) to expand
    v_k* = {v_k | min δ(v_k)}
    P(S) ← (S^1, I^1) using v_k*
    k ← k + 1
end while
return P(S)
```

EMD value of $u_2 = 22 + 11 = 33$, and of $u_3 = 22 + 7 = 29$. Furthermore, we visit edge $\mathfrak{p}u_2, u_3\mathfrak{q}$ first since $w\mathfrak{p}u_2, u_3\mathfrak{q} \quad 22$ is the largest edge weight of u_2. We refine the sense for each equivalence class based on Algorithm 3. Lastly, the algorithm returns the final sense assignment.

Complexity. Computing an initial assignment takes time $\mathrm{O}(|\Pi_X| \quad m)$ to evaluate all $|\Pi_X|$ equivalence classes, and in the worst case, the total m senses. Traversing the dependency graph is in the same order as BFS, taking time $O\mathfrak{p}|V| \quad |E|\mathfrak{q} \quad m\mathfrak{q}$, and to evaluate senses for each visited node.

4.5 Computing Repairs

After each equivalence class x is assigned a sense, we have an interpretation from which to identify errors, and to compute repairs. We describe our repair algorithm that first evaluates ontology repair candidates, i.e., values that are in I but not in S under the chosen sense. Second, assuming a set of ontology repairs, we discuss how the remaining data violations are modeled and repaired.

4.5.1 Ontology Repairs

We consider ontology repairs that add new values v P I, but v R S. Let $Cand$pSq represent all candidate ontology repair values. We iterate through $Cand$pSq by considering repairs of size k, k t$1, 2, ..., |Cand$pSq|u. For ease of presentation, let $\mathcal{V}_k$ represent the set of candidate ontology repairs of size k. For each $\mathcal{V}_k$, we compute the number of data repairs δ_k needed to achieve consistency w.r.t. Σ, and select data repairs with a minimum number of updates for each k. To generate a Pareto-optimal set of repairs, we select pS^1, I^1q that are minimal for each k value.

Generating all k-combinations of solutions has exponential complexity, taking Op$2^{|Cand$pSq|}$q. We propose a greedy strategy to selectively consider the most promising solutions. We use the *beam search* strategy, a heuristic optimization of breadth-first search, that expands the top-b most promising nodes, for a beam size b. The parameter b can be tuned according to application requirements; in our case, we select b by maximizing the probability of selecting the value from a random sequence. The

Secretary Problem addresses this issue, where the objective is to select one secretary from w candidates, and applicants are considered in some random order (all orders are equally likely) [46]. Applicants are chosen immediately after being interviewed, and decisions are non-reversible. Prior work has shown that it is optimal to interview b $\frac{w}{e}$ candidates, where e is the exponential constant. This strategy selects the best candidate with probability $\frac{1}{e}$, with a competitive ratio e. In our case, we set w $|Cand(S)|$.

We organize the candidate repairs as a set-containment lattice, where nodes at level k represent ontology repairs of size k, e.g., at level 1, we consider single value repairs. Let v_k represent a node at level k, i.e., the repairs in $v_k \in \mathcal{V}_k$. Each node v_k is created by augmenting a node at level k 1 with a single repair value. We begin the search at a node v_1 at level k 1. As we visit a node v_k, we compute the minimum number of data repairs for the current candidate ontology repair v_k, i.e., assuming S^1 $(S \cup v_k)$. We select the top-b nodes at each level k with the minimum number of data repairs, to further explore at the next level k 1. We continue this traversal at each subsequent level until we have $I^1 \models \Sigma$ w.r.t. S^1, or until we reach the leaf level. Algorithm 5 presents the details.

4.5.2 Approximating Minimum Data Repairs

Given v_k to derive S^1, we compute the necessary data repairs I^1 such that $dist(I, I^1) \leq \tau$. Computing a minimal number of data repairs to I such that $I^1 \models \Gamma$, for a set of FDs Γ, is known to be NP-hard [65]. Since OFDs subsume FDs, i,e., $\Gamma \subseteq \Sigma$, this

intractability result, unfortunately, carries over to OFDs. Beskales et al., show that the minimum number of data repairs can be approximated by upper bounding the minimum number of necessary cell changes [51]. In our implementation, we adapt their RepairData algorithm that is shown to compute an instance I^1, by cleaning tuples one at a time, such that the number of changed cells is at most $P \quad 2 \min\mathsf{t}|Z|, |\Sigma|\mathsf{u}$, where $|Z|$ is the number of (unique) consequent attributes in Σ, and $|\Sigma|$ denotes the number of OFDs. We seek P-approximate, τ-constrained repairs, where a P-approximate τ-constrained repair $\mathsf{p}S^1, I^1\mathsf{q}$ is a repair in U such that $dist\mathsf{p}I, I^1\mathsf{q} \ \mathsf{¤} \ \tau$, and there is no other repair $\mathsf{p}S^2, I^2\mathsf{q} \ \mathsf{P} \ U$ such that $\mathsf{p}dist\mathsf{p}S, S^2\mathsf{q}, P \quad dist\mathsf{p}I, I^2\mathsf{qq}$ $\mathsf{p}dist\mathsf{p}S, S^1\mathsf{q}, \tau\mathsf{q}$.

We transform the data repair problem to the problem of finding a minimum vertex cover, where nodes represent a tuple t_i, and an edge $\mathsf{p}t_i, t_j\mathsf{q}$ represents that t_i, t_j conflict w.r.t. an OFD. We generate a conflict graph for I, w.r.t. Σ, S^1, and compute a 2-approximate minimum vertex cover of the conflict graph [47]. We annotate the edges in the conflict graph with: (i) the violated OFD ϕ; and (ii) the candidate repair and sense.

Example 15: Table 4.2 shows a subset of Table 4.1 where t_{11}[CTRY] is updated to *United States*. Consider ϕ_1 : [CC] Ñ [CTRY] with an ontological equivalence between *USA*, *America*, and ϕ_2 : [SYMP, DIAG] Ñ [MED] w.r.t. the drug (MED) ontology shown in Figure 4.1. If the *FDA* sense is selected for ϕ_2 (according to Algorithm 4), then Figure 4.7 shows the corresponding conflict graph. I

For each error tuple t_e, we modify attribute A by considering candidate repairs

Table 4.2: Sample clinical trials data.

id	CC	CTRY	SYMP	DIAG	MED
t_8	US	USA	headache	hypertension	cartia
t_9	US	USA	headache	hypertension	ASA
t_{10}	US	America	headache	hypertension	tiazac
t_{11}	US	Uni. States	headache	hypertension	adizem

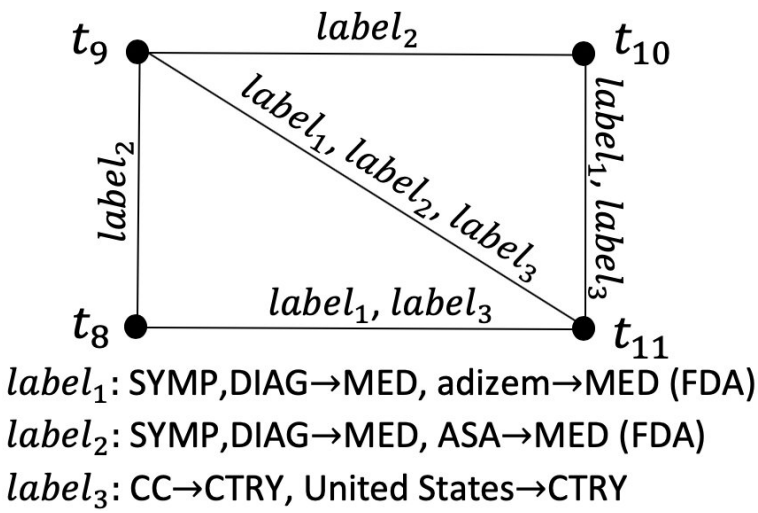


Figure 4.7: Conflict graph for Table 4.2.

Table 4.3: Sample repairs.

Ont. Repair value (sense)	$\lvert S, S^1 \rvert$	Conflict Edges	C_{2opt}	δ_P
H	0	$\mathsf{p}t_8, t_9\mathsf{q}$, $\mathsf{p}t_8, t_{11}\mathsf{q}$, $\mathsf{p}t_9, t_{10}\mathsf{q}$ $\mathsf{p}t_9, t_{11}\mathsf{q}$, $\mathsf{p}t_{10}, t_{11}\mathsf{q}$	t_9, t_{11}	4
ASA (FDA)	1	$\mathsf{p}t_8, t_{11}\mathsf{q}$, $\mathsf{p}t_9, t_{11}\mathsf{q}$, $\mathsf{p}t_{10}, t_{11}\mathsf{q}$	t_{11}	2
adizem (FDA)	1	$\mathsf{p}t_8, t_9\mathsf{q}$, $\mathsf{p}t_8, t_{11}\mathsf{q}$, $\mathsf{p}t_9, t_{10}\mathsf{q}$ $\mathsf{p}t_9, t_{11}\mathsf{q}$, $\mathsf{p}t_{10}, t_{11}\mathsf{q}$	t_9, t_{11}	4
United States	1	$\mathsf{p}t_8, t_9\mathsf{q}$, $\mathsf{p}t_8, t_{11}\mathsf{q}$, $\mathsf{p}t_9, t_{10}\mathsf{q}$ $\mathsf{p}t_9, t_{11}\mathsf{q}$, $\mathsf{p}t_{10}, t_{11}\mathsf{q}$	t_9, t_{11}	4
adizem (FDA) United States	2	$\mathsf{p}t_8, t_9\mathsf{q}$, $\mathsf{p}t_9, t_{10}\mathsf{q}$, $\mathsf{p}t_9, t_{11}\mathsf{q}$	t_9	2
...	...	...	...	...

from the domain of A when there is sufficient evidence to do so, or to a value v^1 P S^1 such that the values tv^1, au are synonyms for all a P Π_X. As each t_e is repaired, we remove its corresponding nodes and edges, and re-generate the conflict graph in linear time. The algorithm continues until all tuples t_e have been removed from the conflict graph, and I^1 is returned. The algorithm runs in Op$|E|$ $|I|$q, where $|E|$ is the number of edges in the conflict graph [51].

Example 16: Table 4.3 shows an execution of OFDClean listing: (i) candidate ontology repairs (value, sense); (ii) number of ontology repairs $|\Delta$pS, S^1q$|$; (iii) conflicts in the conflict graph (Figure 4.7); (iv) C_{2opt}: the 2-approximation vertex cover of the conflict graph, indicating the tuples to be repaired; and (v) δ_PpS^1, Iq: the upper bound on the minimum number of data repairs to I. Consider an ontology repair to add ASA to S, which would eliminate edges pt_8, t_9q, pt_9, t_{10}q, and leave only tuple t_{11} to be updated with at most two minimum data repairs. Comparing all single candidate ontology repairs, this repair is the minimum, involving one and two, ontology and data repairs, respectively. |

4.6 Experiments

In this section, we study the accuracy and performance of OFDClean. We show the effectiveness of our sense selection algorithm, and compare OFDClean against the HoloClean [83] repair algorithm and demonstrate the benefits of ontology repair. We evaluate the scalability of OFDClean by varying the beam size, error rates, and number

of tuples.

4.6.1 Experimental Setup

Experiments were performed using four Intel Xeon processors at 3.0GHz each with 32GB of memory. Algorithms were implemented in Java. The reported runtimes are averaged over six executions.

Datasets. We use two real datasets, and the U.S. National Library of Medicine Research and WordNet ontologies [77].

Kiva [64] describes loans issued over two years via the Kiva.org online crowdfunding platform to financially excluded citizens around the world. There are 670K records and 15 attributes, including loan principal amount, loan activity, country code, country, region, funded time and usage.

Clinical [56]: The Linked Clinical Trials (LinkedCT.org) database describes clinical patient data characteristics such as the clinical study, country, medical diagnosis, drugs, illnesses, symptoms, treatment, and outcomes. We use a portion of the dataset with 250K records and 15 attributes.

We use real ontologies to ensure coverage of the (RHS) attribute domain, e.g., the Medical Research ontology covers values in the DRUG attribute of the clinical trials data, while considering that some medications have different names across different countries (senses). We aim to maximize coverage (upwards of 90%+ coverage for some attributes). Values not covered by an ontology are candidates for ontology repair.

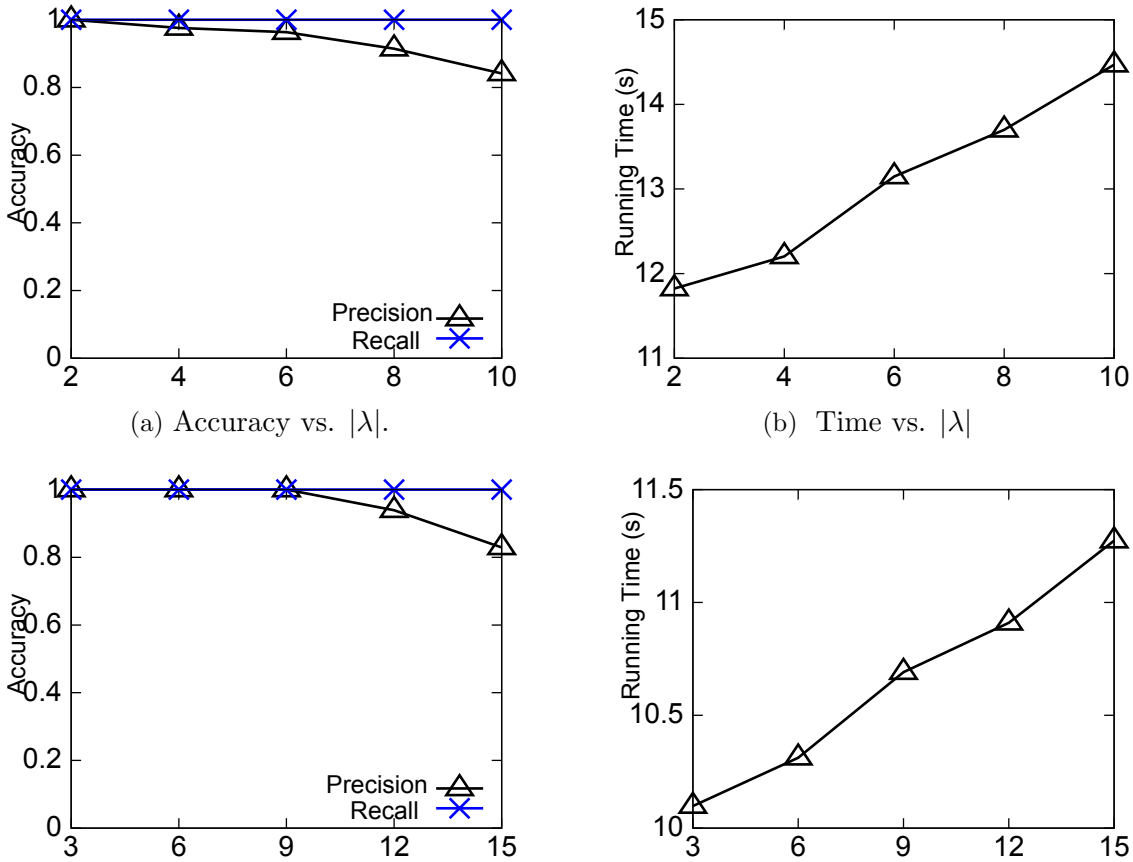
Table 4.4: Parameter values (defaults in bold)

Symbol	**Description**	**Values**
$\|\lambda\|$	# senses	2, **4**, 6, 8, 10
$err\%$	error rate	**3**, 6, 9, 12, 15
N	# tuples (Million)	0.2, **0.4**, 0.6 0.8, 1 (clinical)
b	beam size	1, 2, **3**, 4, 5
$inc\%$	incompleteness rate	2, **4**, 6, 8, 10
$\|\Sigma\|$	# OFDs	**10**, 20, 30, 40, 50

Parameters and Ground Truth. Table 4.4 shows the parameter values and their defaults. We inject errors randomly into the consequent attributes. Errors are inserted by either changing an existing value to a new value (not in the attribute domain), or to an existing domain value. We consider the original data values as the ground truth, as we vary the error rate $err\%$. We specify τ as a percentage (100% allows the algorithm to freely change the data). We set τ 65% to balance similarity between I and I^1, and flexibility to consider new values via ontology repairs.

Comparative Techniques.

HoloClean [83] provides holistic data repair by combining multiple input signals (integrity constraints, external dictionaries, and statistical profiling), and uses probabilistic inference to infer dirty values. We input to HoloClean: (i) a set of denial constraints translated from the given dependencies; (ii) external reference sources such as the National Drug Code Directory [95]; and (iii) we profile the data to obtain statistical frequency distributions of each attribute's domain. We compare OFDClean against HoloClean since it also considers external information during data repair.



(a) Accuracy vs. $|\lambda|$.

(b) Time vs. $|\lambda|$

(c) Accuracy vs. $err\%$

(d) Time vs. $err\%$

Figure 4.8: Sense assignment effectiveness (Clinical dataset).

4.6.2 Sense Selection Performance

We now use the clinical dataset to evaluate the performance of our sense selection algorithm.

Exp-6: Vary $|\lambda|$. Figure 4.8a shows the accuracy as we increase the number of senses $|\lambda|$. Since every equivalence class is assigned a sense, we achieve 100% recall, independent of the number of senses. Precision decreases with more senses due to

more available choices, but still remains above 80%. As we evaluate more senses, the runtimes increase linearly, as shown in Figure 4.8b.

Exp-7: Vary $err\%$**.** Figure 4.8c shows that precision declines linearly as the number of errors increases. It becomes more challenging to select the correct sense when multiple senses contain overlapping (erroneous) values. Figure 4.8d shows that runtimes increase as more candidate senses and refinements have to be evaluated to align distributional deviations caused by an increasing number of errors.

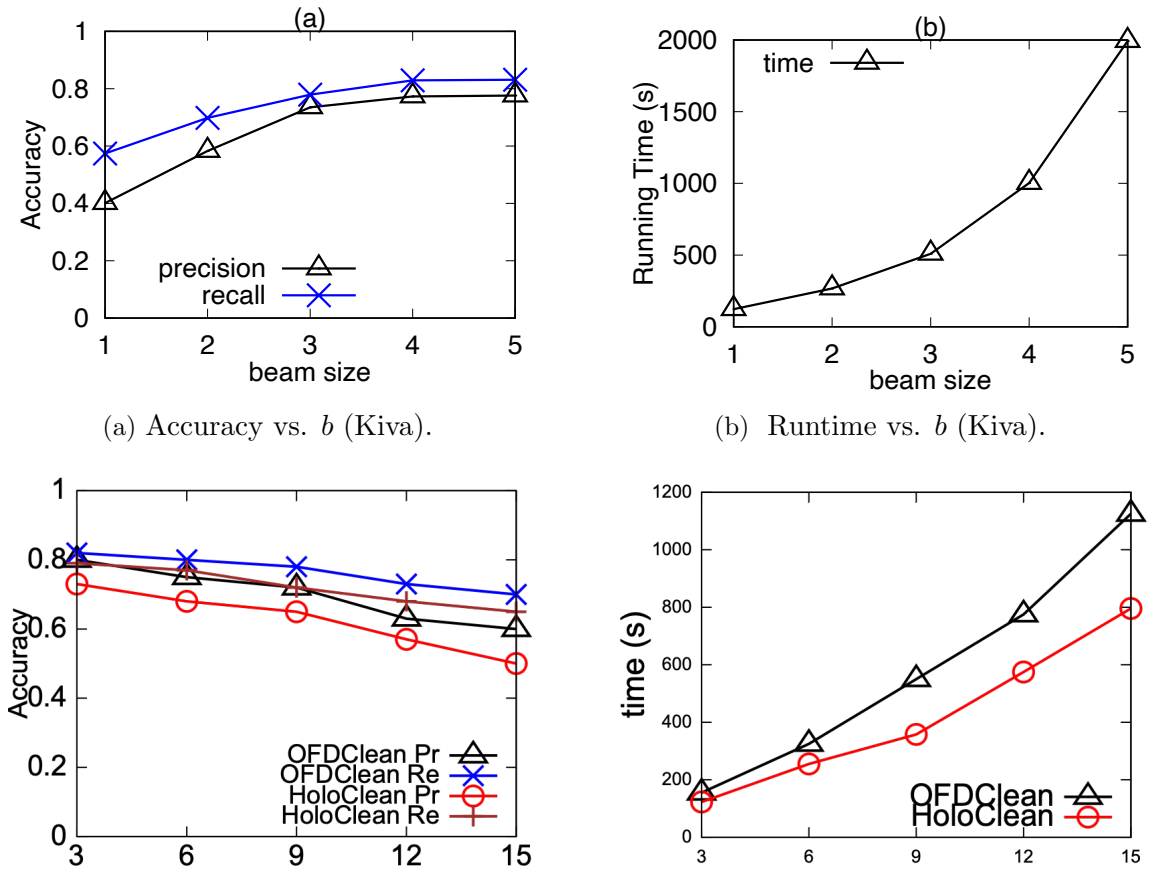
Exp-8: Vary N**.** We increase the number of tuples in the Clinical dataset up to 1M records. This increases the number of equivalence classes, but our sense assignment strategy achieves over 90% precision. We found that the increase in N did not impact the precision and recall accuracy (figure omitted for brevity). With more equivalence classes, there is an increased likelihood of overlap, leading to longer runtimes to resolve shared errors, as shown in Table 4.5.

Table 4.5: Varying N runtimes.

N (M)	**0.2**	**0.4**	**0.6**	**0.8**	**1**
(sec)	9.3	11.8	17.1	23.8	27.2

4.6.3 OFDClean Performance

Exp-9: Vary beam size b**.** Figure 4.9a shows that as we evaluate more candidate ontology repairs, we achieve higher precision and recall. The incremental benefit



(a) Accuracy vs. b (Kiva).

(b) Runtime vs. b (Kiva).

(c) Acc. vs. err% (Kiva).

(d) Time vs. err% (Kiva).

Figure 4.9: OFDClean Performance.

becomes marginal once we have found the best repair, as reflected between b 4 and b 5. Figure 4.9b shows that the runtime increases exponentially due to the number of repair combinations that must be evaluated for increasing b. To manage runtime costs in practice, the beam size can be tuned according to accuracy requirements, with initial tuning guidelines given in Section 4.5.1.

Exp-10: Vary *err%*. For increasing error rates, Figure 4.9c shows that accuracy declines due to overlapping values between antecedent and consequent values among

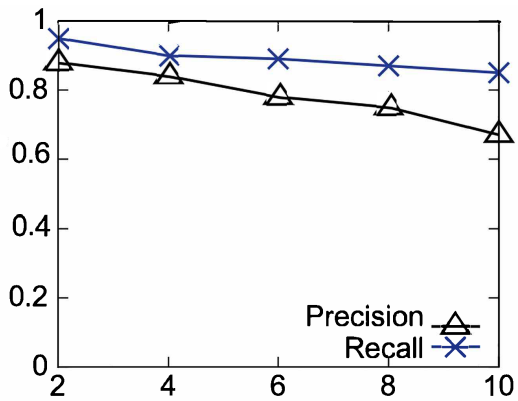


Figure 4.10: Acc. vs. *inc*%

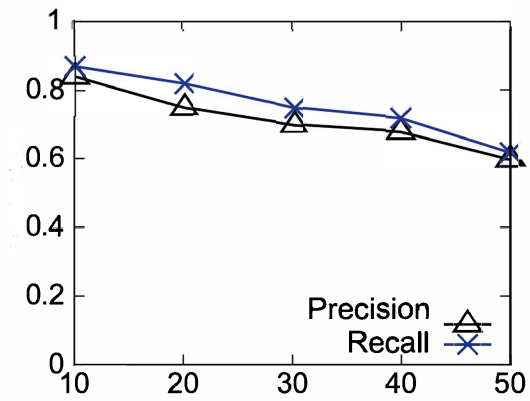


Figure 4.11: Acc. vs. |Σ|.

multiple OFDs. An update to attribute A P X_2 for $\phi_1 : X_1$ Ñ A, $\phi_2 : X_2$ Ñ B, changes the distribution of equivalence classes w.r.t. X_2, leading to lower recall and precision. Figure 4.9d shows that runtimes increase as more errors are evaluated, and a larger space of repairs must be considered.

Exp-11: Vary *inc*%**.** Figure 4.10 shows the repair accuracy as we vary the incompleteness rate, *inc*%, which measures the percentage of values in I, but not in S, where such errors are resolved via ontology repairs. As *inc*% increases, the precision declines, as some repair values are added to the wrong sense. Recall scores are more consistent, achieving above 85%, with slight linear decline as error values are corrected by data updates.

Exp-12: Vary |Σ|. Figure 4.11 shows that increasing the number of OFDs causes both precision and recall to decline as an increasing number of attributes overlap among the OFDs. OFDClean resolves errors when attribute overlap occurs in the consequent attribute between two OFDs, i.e., $\phi_1 : X_1$ Ñ A_1, $\phi_2 : X_2$ Ñ A_2, $A_1 \quad A_2$. However, when errors arise w.r.t. ϕ_1 due to an update in attribute A_2 P X_1, we do not

re-evaluate ϕ_1 again. We intend to explore this increased space of repairs in future work.

Exp-13: Vary N. Increasing the number of tuples, N, does not significantly impact the repair accuracy, with an average variation of 1.4% in precision (figure omitted for brevity). Similarly, OFDClean achieves linear runtime performance, as shown in Table 4.6. This runtime increase occurs due to the larger number of equivalence classes created by newly inserted domain values, and the overhead of evaluating ontology repair combinations.

Table 4.6: OFDClean runtimes (m)

N (K)	**50**	**100**	**150**	**200**	**250**
	166	175	182	198	217

Exp-14: Comparative Discussion. To the best of our knowledge, there are no ontology repair algorithms coupled with data repair, especially for sense selection. We compare OFDClean against HoloClean [83], which considers external information during repair selection. Figure 4.9c shows that OFDClean achieves increased precision and recall over HoloClean by 7.4% and 4.4%, respectively. By recognizing the same interpretation of different values via a sense, OFDClean selectively identifies errors and improves precision. This comes at a cost in runtime (Figure 4.9d) since OFDClean must evaluate a space of ontology repairs to find those that minimize the number of data repairs.

4.7 Related Work

Data Dependencies. Our work is similar to ontological dependencies and relaxed notions of FDs. Motik et al. define integrity constraints using the Web Ontology Language (OWL), where OWL ontologies are often incomplete [76]. They propose an extension of OWL to validate ontology completeness by defining inclusion dependencies and domain constraints to check for missing and valid domain values within an ontology.

Ontological Graph Keys (OGKs) exploit ontologies to enhance keys for graphs [72]. An OGK incorporates an event pattern defined on an entity, which identifies instances w.r.t. concept similarity in an ontology S. OGKs characterize entity equivalence using ontological similarity. Our work shares a similar spirit to leverage ontological relationships and concepts. However, existing techniques do not consider the notion of senses to enable multiple interpretations of values in an ontology.

Constraint-based Data Cleaning. FDs are used as a benchmark to propose data repairs such that the data and the FDs are consistent [17, 82, 24]. Extensions have relaxed the notion of equality in FDs, and used similarity and matching functions to identify errors, and propose repairs [15, 55]. While our work is similar in spirit, we differ in the following ways: (1) the similarity functions match values based only on syntactic string similarity; and (2) the notion of senses is not considered, and similarity under multiple interpretations is not possible.

Holistic Data Cleaning. Probabilistic approaches have shown promise to tackle the multiple facets of data cleaning. ERACER [74] and HoloClean [83] learn from

given attribute templates and training samples to infer clean values. Existing systems either use a broad set of constraints [48], or bound repairs according to maximum likelihood [99]. These techniques capture context either via external sources or additional statistics. We advocate for a deeper integration of context into the data cleaning framework so that such "inconsistencies" are not flagged in the first place. To the best of our knowledge, the only prior work on using ontologies for data cleaning is KATARA [25], which includes simple patterns in ontologies such as "France" hasCapital "Paris", and does not integrate ontologies into the definition of integrity constraints. Context takes a central role in BARAN, which defines a set of error corrector models with a context-aware data representation to improve precision [73]. Incorporating these models into OFDClean and resolving conflicts between contextual models and ontologies are interesting avenues of future work.

Knowledge Base Incompleteness. Computational fact checking against Knowledge Bases (KBs) has emerged in recent years to automatically verify facts from different domains. The main problem in fact checking with KBs is that the reference information may be incomplete. That is, under an Open World Assumption (OWA), we cannot assume that all facts in a KB are complete; facts not in a KB may be false or just missing [60]. In our work, we do not make such open assumptions. To improve KB quality, the RuDiK system discovers positive rules to mitigate incompleteness (e.g., if two persons have the same parent, then they are siblings), and negative rules to detect errors (e.g., if two persons are married, one cannot be a child of the other) [79]. Incorporating such inference into OFDClean as a new ontology repair operation will enrich existing ontological relationships.

4.8 Conclusions

Ontology Functional Dependencies (OFDs) capture domain relationships found in ontologies. We study the repair problem with respect to a set of OFDs, and propose an algorithm to find a sense for each equivalence class to provide an interpretation for each OFD against the data. Our proposed framework, OFDClean, combines sense selection with ontology and data repair such that the total number of updates to the data and to the ontology is minimized. Our experiments showed that our algorithms are effective, as OFDs are useful data quality rules to capture domain relationships, and significantly reduce the number of false positive errors in data cleaning solutions that rely on traditional FDs. As next steps, we intend to explore cleaning with respect to inheritance OFDs, which represent hierarchical relationships among data values.

Chapter 5

Conclusion and Future Work

In this thesis, we tackled various challenges associated with two data quality dimensions, namely, data currency and data consistency. For the challenges in the data currency dimension, we propose two quantitative approaches: (1) we present a probabilistic system for identifying and cleaning stale values by learning the spatio-temporal update patterns in an update history, which outperforms four existing approaches by an average +34% recall to identify stale cells, and +28% F1-score in repair accuracy; (2) We derive a theoretical bound on the confidence that a replica is current or stale in distributed database settings. We implement a novel consistency level DYNAMIC in Cassandra, and the experiments show the effectiveness and efficiency of our approach. For the challenges in the data consistency dimension, we present OFDClean, an algorithm that computes a minimal number of modifications to an ontology and to the data to satisfy a given set of OFDs. Compared to HoloClean, a holistic data cleaning framework, we achieve increased precision and recall by +7.4% and +4.4%,

respectively.

CurrentClean. 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, with improved error detection and repair accuracy over state-of-the-art techniques.

Replica Currency Estimation. We provide theoretical bounds on the confidence of replica currency estimation. Our system computes with a minimum probability p, whether the retrieved replicas are current or stale. Using this confidence-bounded replica currency estimation, we implement a novel DYNAMIC consistency level in the open-source, NoSQL database, Cassandra. Experiments show that the proposed replica currency estimation is effective and efficient. In most tested scenarios, with various query loads and cluster configurations, we show our estimations with confidence levels of at least 0.99. Moreover, the overheads introduced due to estimation scoring and training are low, incurring only 0.76% to 1.17% of the original query processing and replica synchronization time costs, respectively.

OFDClean. We present OFDClean, an algorithm that computes a minimal number of modifications to an ontology and to the data to satisfy a given set of OFDs. To consider the possible interpretations of the data, OFDClean selects the best interpretation (sense) for an equivalence class of tuples, such that the selected sense minimizes

the number of required modifications. We evaluate the performance and effectiveness of OFDClean using real datasets over varying parameter values, and show that OFDs can reduce the number of false positive errors in data cleaning techniques that rely on traditional FDs.

5.1 Future Work

Naturally there is more work to be done. While we study the problems of improving data consistency and estimating data currency, we have not studied the relationship between these two important dimensions.

To encourage more future study in this area, we can consider creating a taxonomy of techniques, similar to data cleaning, where declarative and quantitative approaches are possible. Currency constraints [39]) are declarative specifications with a set of data quality rules, and the data is stale when a rule is violated. In quantitative solutions, such as PBS [8]), often relies on a statistical model to infer the probability of a cell being stale.

Secondly, a more in-depth study of the similarities and relationships between the requirements and techniques to achieve these two quality dimensions would be a next step. Data currency focuses on identifying and repairing stale values for a given entity with different versions of values. Data consistency is often measured by a set of data quality rules, and inconsistent data may exist across multiple entities. However, cleaning stale data helps repair inconsistent data and vice versa [39]. For

instance, CurrentClean considers the values to be stale if they violate the spatio-temporal update patterns that are learned in the update history, and these stale values are inconsistent if we consider update patterns as a set of data quality rules. On the other hand, the violations of OFDs may occur as data and ontology changes over time and the ontology is stale. By considering the update patterns and update value correlations in the update history of data and ontology, CurrentClean is another approach to resolving the violations w.r.t. OFDs.

5.1.1 CurrentClean

In future work, (1) we plan to relax the assumption that the history of updates, H, is clean, by considering that H contains a proportion of errors, which makes the problem harder since we must now balance the trade-offs between data currency and data accuracy; (2) we are exploring extensions to the model to consider learning and inference beyond the adjacent time unit to multiple time units to capture more complex temporal dependencies; (3) the size of a time unit (δ) is a model parameter that influences the accuracy of currency estimation. The optimal value of δ is dependent on the frequency of queries in the update history. Within a time unit, the UModel and RModel do not distinguish among these updates nor do they consider the update frequency of a cell within a time interval. Each model selects the most recent update (or updated value) to use as part of the learning and inference. This leads to a large proportion of errors not being identified and incorrectly repaired since δ is not selected properly. We are addressing this limitation as a next step, to differentiate among multiple updates to a cell within a time unit; (4) CurrentClean may

inaccurately identify stale cells when there exists update distribution bias in H, e.g., H is extreme incomplete. To decrease potential bias in H, a possible approach is to augment the incomplete H with an open source data that has a similar update distribution, to produce a relative complete update history based on a data augmentation technique.

5.1.2 Replica Currency Estimation

In this work, we assumed a fixed timestamp of a replica synchronization operation e.g., regularly by anti-entropy or occasionally by read repair in Cassandra. We intend to study in future work how this synchronization time can be adaptive to the replica response times. In addition, we believe that the replica currency estimation and the DYNAMIC consistency level are extensible to other (NoSQL) databases, such as HBase, and intend to implement such features as future work. Lastly, we bound the estimation confidence based on a machine learning model's generalization error, specifically, we use linear regression model which provides a lower bound. In the next step, we plan to study whether tighter bounds are possible under specific scenarios.

5.1.3 OFDClean

As next steps, we intend to (1) explore cleaning with respect to inheritance OFDs, which represent hierarchical relationships among data values; (2) relax the assumption of no overlapping attributes between antecedent and consequent attributes among

multiple OFDs; (3) incorporate human-in-the-loop models that leverage domain expertise to target and reduce the space of senses considered. For example, by looking at the data distributions w.r.t. specific OFDs, we can reduce the space of candidate senses that must be evaluated. In addition, incorporating human-in-the-loop feedback to select the most desirable repairs from our set of Pareto-optimal repairs is another avenue of future study.

Bibliography

[1] Crowdflower 2016 data science report. https://www.kdnuggets.com/2016/04/crowdflower-2016-data-science-repost.html, 2016.

[2] Z. Abedjan, C. G. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker. Temporal rules discovery for web data cleaning. *PVLDB*, 9(4):336–347, 2015.

[3] F. Akal, C. Türker, H.-J. Schek, Y. Breitbart, T. Grabs, and L. Veen. Fine-grained replication and scheduling with freshness and correctness guarantees. In *International Conference on Very Large Data Bases*, pages 565–576, 2005.

[4] M. Alipour, Z. Zheng, F. Chiang, L. Golab, and J. Szlichta. Contextual data cleaning. *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 21–24, 2018.

[5] Apache Cassandra. Open Source NoSQL Database. https://cassandra.apache.org/.

[6] Apache HBase. Welcome to Apache HBase. https://hbase.apache.org/.

[7] Apoorv Patne. Extracted bank account statements of various bank accounts. https://www.kaggle.com/apoorvwatsky/bank-transaction-data, 2019.

[8] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endow.*, 5(8):776–787, 2012.

[9] S. Baskaran, A. Keller, F. Chiang, L. Golab, and J. Szlichta. Efficient discovery of ontology functional dependencies. *Technical report, CoRR, https://arxiv.org/abs/1611.02737*, 12 pages, 2016.

[10] S. Baskaran, A. Keller, F. Chiang, L. Golab, and J. Szlichta. Efficient discovery of ontology functional dependencies. In *CIKM*, pages 1847–1856, 2017.

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

[12] I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou. Patient subtyping via time-aware LSTM networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,*, pages 65–74. ACM, 2017.

[13] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[14] L. Berti-Equille, T. Dasu, and D. Srivastava. Discovery of complex glitch patterns: A novel approach to quantitative data cleaning. In *ICDE*, pages 733–744, 2011.

[15] L. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query

answering with matching dependencies and matching functions. In *ICDT*, pages 268–279, 2011.

[16] T. Bleifuß, L. Bornemann, T. Johnson, D. V. Kalashnikov, F. Naumann, and D. Srivastava. Exploring change - A new dimension of data analytics. *PVLDB*, 12(2):85–98, 2018.

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

[18] E. A. Brewer. Towards robust distributed systems (abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, page 7, 2000.

[19] Q. Cai, Z. Xie, M. Zhang, G. Chen, H. V. Jagadish, and B. C. Ooi. Effective temporal dependence discovery in time series data. *PVLDB*, 11(8):893–905, 2018.

[20] Y. Cao, W. Fan, and W. Yu. Determining the relative accuracy of attributes. In *SIGMOD*, pages 565–576, 2013.

[21] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data. In *7th Symposium on Operating Systems Design and Implementation (OSDI '06)*, pages 205–218, 2006.

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

[23] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457, 2011.

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

[25] 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*, pages 1247–1261, 2015.

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

[27] J. Cipar, G. Ganger, K. Keeton, C. B. Morrey, C. A. Soules, and A. Veitch. Lazybase: Trading freshness for performance in a scalable database. In *European Conference on Computer Systems*, pages 169–182, 2012.

[28] S. Clemens. 7 facts about data quality. *InsightSquared*, 2012.

[29] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, pages 315–326, 2007.

[30] C. Cuadras and C. Arenas. A distance based regression model for prediction with mixed data. *Communications in Statistics-Theory and Methods*, 19(6):2261–2279, 1990.

[31] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: A commodity data cleaning system. In *SIGMOD*, pages 541–552, 2013.

[32] T. Dasu and J. M. Loh. Statistical distortion: Consequences of data cleaning. *PVLDB*, 5(11):1674–1683, 2012.

[33] A. H. de Souza Júnior, F. Corona, G. D. A. Barreto, Y. Miché, and A. Lendasse. Minimal learning machine: A novel supervised distance-based approach for regression and classification. *Neurocomputing*, 164:34–44, 2015.

[34] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, pages 205–220, 2007.

[35] P. J. Denning. The locality principle. *Communications of the ACM*, 48(7):19–24, 2005.

[36] X. L. Dong, A. Kementsietsidis, and W.-C. Tan. A time machine for information: Looking back to look forward. *SIGMOD Rec.*, 45(2):23–32, 2016.

[37] W. Fan. Constraint-driven database repair. pages 458–463. Springer US, 2009.

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

[39] W. Fan, F. Geerts, N. Tang, and W. Yu. Inferring data currency and consistency for conflict resolution. In *ICDE*, pages 470–481, 2013.

[40] W. Fan, F. Geerts, N. Tang, and W. Yu. Conflict resolution with data currency and consistency. *J. Data and Information Quality*, 5(1–2), 2014.

[41] W. Fan, F. Geerts, and J. Wijsen. Determining the currency of data. In *PODS*, pages 71–82, 2011.

[42] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, 3(1):173–184, 2010.

[43] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *SIGMOD*, pages 469–480, 2011.

[44] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *VLDB J.*, 21(2):213–238, 2012.

[45] F. Fatemi. Best practices for data hygiene. *Forbes*, 2019.

[46] T. S. Ferguson. Who solved the secretary problem? *Statistical Science*, 4(3):294–296, 1989.

[47] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. 1979.

[48] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.

[49] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*. CRC press, 2013.

[50] L. George. *HBase - The Definitive Guide: Random Access to Your Planet-Size Data*. O'Reilly, 2011.

[51] L. Golab, I. F. Ilyas, G. Beskales, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552, 2013.

[52] I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[53] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. *In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 173–182, 1996.

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

[55] S. Hao, N. Tang, G. Li, J. He, N. Ta, and J. Feng. A novel cost-based model for data repairing. *TKDE*, 29(4):727–742, 2017.

[56] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang. Linkedct: A linked data space for clinical trials. Technical Report CSRG-596, University of Toronto, 2009.

[57] M. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Trans. Database Syst.*, 12(2):170–194, 1987.

[58] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[59] Hristo Mavrodiev. London bike sharing dataset. `https://www.kaggle.com/hmavrodiev/london-bike-sharing-dataset`, 2019.

[60] V. Huynh and P. Papotti. Towards a benchmark for fact checking with knowledge bases. In *Companion Proceedings of the The Web Conference 2018*, page 1595–1598. International World Wide Web Conferences Steering Committee, 2018.

[61] Intelligent Data Analytics Lab @ MSU. Time-Aware Long-Short Term Memory. `https://github.com/illidanlab/T-LSTM`.

[62] H. V. Jagadish, Koudas, Muthukrishnan, Poosala, and Suel. Optimal histograms with quality guarantees. In *VLDB*, pages 275–286, 1998.

[63] A. Johnson, T. Pollard, M. Ghassemi, B. Moody, P. Szolovits, and R. Mark. MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3(160035), 2016.

[64] Kiva. Data science for good: Kiva crowdfunding. `https://www.kaggle.com/kiva/data-science-for-good-kiva-crowdfunding?select=loan_themes_by_region.csv`.

[65] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

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

[67] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *ICDE*, pages 1275–1278, 2009.

[68] A. Labrinidis and N. Roussopoulos. Exploring the tradeoff between performance and data freshness in database-driven web servers. *The VLDB Journal*, 13(3):240–255, 2004.

[69] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *Operating Systems Review*, 44(2):35–40, 2010.

[70] B. Lowerre. The harpy speech recognition system. *Ph.d. Thesis, Carnegie-Mellon University*, 1976.

[71] H. Lu, K. Veeraraghavan, P. Ajoux, J. Hunt, Y. Song, W. Tobagus, S. Kumar, and W. Lloyd. Existential consistency: measuring and understanding consistency at facebook. *In Proceedings of the 25th Symposium on Operating Systems Principles*, pages 295–310, 2015.

[72] H. Ma, M. Alipourlangouri, Y. Wu, F. Chiang, and J. Pi. Ontology-based entity matching in attributed graphs. *Proc. VLDB Endow.*, 12(10):1195–1207, 2019.

[73] M. Mahdavi and Z. Abedjan. Baran: Effective error correction via a unified context representation and transfer learning. *Proc. VLDB Endow.*, 13(11):1948–1961, 2020.

[74] C. Mayfield, J. Neville, and S. Prabhakar. ERACER: a database approach for statistical inference and data cleaning. In *SIGMOD*, pages 75–86, 2010.

[75] M. Milani, Z. Zheng, and F. Chiang. Currentclean: Spatio-temporal cleaning of stale data. *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 172–183, 2019.

[76] B. Motik, I. Horrocks, and U. Sattler. Adding Integrity Constraints to OWL. In *OWL: Experiences and Directions (OWLED)*, 2007.

[77] National Library of Medicine. Biomedical terminology, ontologies. `https://mor.nlm.nih.gov`, 2016.

[78] Open Sourcing Mental Illness, LTD. Mental health in tech survey. `https://www.kaggle.com/osmi/mental-health-in-tech-survey`, 2016.

[79] S. Ortona, V. Meduri, and P. Papotti. Robust discovery of positive and negative rules in knowledge bases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1168–1179, 2018.

[80] E. Pacitti, E. Simon, and R. Melo. Improving data freshness in lazy master schemes. In *International Conference on Distributed Computing Systems*, pages 164–171, 1998.

[81] D. Potts and C. Sammut. Incremental learning of linear model trees. *Mach. Learn.*, 61(1-3):5–48, 2005.

[82] N. Prokoshyna, J. Szlichta, F. Chiang, R. Miller, and D. Srivastava. Combining quantitative and logical data cleaning. *PVLDB*, 9(4):300–311, 2015.

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

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

[85] Carsten. 400+ crypto currency pairs at 1-minute resolution (from the Bitfinex exchange including Bitcoin). `https://www.kaggle.com/tencars/392-crypto-currency-pairs-at-minute-resolution`, 2020.

[86] He Group, University of Michigan Medical School. The drug ontology. `http://www.ontobee.org/ontology/DRON`, 2022.

[87] P. J. Rousseeuw and C. Croux. Alternatives to the median absolute deviation. *Journal of the American Statistical Association*, 88(424):1273–1283, 1993.

[88] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vision*, 40(2):99–121, 2000.

[89] M. J. F. K. G. Sanjay Krishnan, Jiannan Wang and T. Kraska. Stale view cleaning: Getting fresh answers from stale materialized views. *Proc. VLDB Endow.*, 48(12):1370–1381, 2015.

[90] J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré. Incremental knowledge base construction using DeepDive. *PVLDB*, 8(11):1310–1321, 2015.

[91] S. Song, A. Zhang, J. Wang, and P. S. Yu. SCREEN: Stream data cleaning under speed constraints. In *SIGMOD*, pages 827–841, 2015.

[92] Y. Sun, Z. Zheng, S. Song, and F. Chiang. Confidence bounded replica currency estimation. *In Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, 2022.

[93] J. Szlichta, P. Godfrey, and J. Gryz. Fundamentals of Order Dependencies. *PVLDB*, 5(11):1220–1231, 2012.

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

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

[96] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *ICDE*, pages 244–255, 2014.

[97] X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *SIGMOD*, pages 1231–1245, 2015.

[98] M. J. Wooldridge. *Foundations of Machine Learning*. MIT Press,, 2012.

[99] 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*, pages 553–564, 2013.

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

[101] T. Yamashita. Distributed view divergence control of data freshness in replicated database systems. *IEEE Transactions on Knowledge and Data Engineering*, 21(10):1403–1417, 2009.

[102] R. Yossi, T. Carlo, and J. Leonidas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 2(40):99–121, 2000.

[103] Yuan Wei, S. H. Son, and J. A. Stankovic. Maintaining data freshness in distributed real-time databases. In *Euromicro Conference on Real-Time Systems*, pages 251–260, 2004.

[104] A. Zhang, S. Song, and J. Wang. Sequential data cleaning: A statistical approach. In *SIGMOD*, pages 909–924, 2016.

[105] A. Zhang, S. Song, J. Wang, and P. S. Yu. Time series data cleaning: From anomaly detection to anomaly repairing. *PVLDB*, 10(10):1046–1057, 2017.

[106] H. Zhang, Y. Diao, and N. Immerman. Recognizing patterns in streams with imprecise timestamps. *PVLDB*, 3(1-2):244–255, 2010.

[107] Z. Zheng. CurrentClean: Datasets and extended experimental results. `http://www.cas.mcmaster.ca/~zhengz13/currentclean.html`, 2019.

[108] Z. Zheng. Contextual data cleaning with ontology FDs. *In Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, pages 2911–2913, 2021.

[109] Z. Zheng, M. Alipour, Z. Qu, I. Currie, F. Chiang, L. Golab, and J. Szlichta. Fastofd: Contextual data cleaning with ontology functional dependencies. *21st International Conference on Extending Database Technology (EDBT)*, pages 694–697, 2018.

[110] Z. Zheng, T. M. Quach, Z. Jin, F. Chiang, and M. Milani. Currentclean: Interactive change exploration and cleaning of stale data. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, page 2917–2920, 2019.

[111] Z. Zheng, L. Zheng, M. Alipour, F. Chiang, L. Golab, J. Szlichta, S. Baskaran, and A. Keller. Contextual data cleaning with ontology functional dependencies. *ACM Journal of Data and Information Quality (JDIQ)*, 2022.