Incremental Schema Recovery

Poonam Kumari
University at Buffalo, SUNY
poonamku@buffalo.edu

Gourab Mitra
University at Buffalo, SUNY
gourabmi@buffalo.edu

Oliver Kennedy
University at Buffalo, SUNY
okennedy@buffalo.edu

ABSTRACT

Abstract goes here

KEYWORDS

Stuff

ACM Reference Format:
Poonam Kumari, Gourab Mitra, and Oliver Kennedy. 1997. Incremental Schema Recovery. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97)*. ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.475/123_4

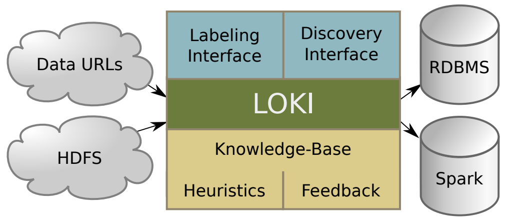

Figure 1: System Overview

1 INTRODUCTION

It's a story as old as time: A student gathers data, makes a graph with the data, writes a paper about the data. Then the student graduates and the data languishes, without so much as a wiki page or README file documenting it. The next student to use the data needs to spend hours, days, or even weeks reverse-engineering it. Then they also graduate and the whole process can start over again.

As a way to break this tragic cycle of data abandonment, we propose *Label Once, and Keep It* (**LOKI**), a data-ingest middleware for incremental, re-usable schema recovery. **LOKI** allows users to assemble schemas on-demand, both (re-)discovering and incrementally refining schema definitions in response to changing data needs. To accomplish this, **LOKI** is built around a knowledge-base of both approximate, as well as exact schema labelings. First, approximate labelings derived from existing open-data sets, user-feedback, and expert-provided heuristics, jump-start the labeling process. When a user first points **LOKI** at a new tabular data set, **LOKI** provides users with a preliminary, default schema. As users confirm and/or override parts of the proposed schema, **LOKI** preserves the labels for the dataset's next user.

In this paper, we detail on our initial efforts to prime the **LOKI** knowledge-base with existing governmental open-data. Specifically...

2 SYSTEM DESIGN

The overall goal of **LOKI** is to streamline the process of developing schemas for existing unlabeled or poorly labeled data sets. As illustrated in Figure 1, **LOKI** lives alongside an existing RDBMS or Spark deployment, and takes as input tabular data in the form of a URL or HDFS file path. **LOKI** provides users with two modes of interaction: (1) A *labeling* interface that assists users in assigning names to existing columns of data, and (2) A *discovery* interface that helps users to search for columns representing particular concepts of interest. Both interfaces are supported by a knowledge-base that combines expert-provided heuristics, learned characteristics, as well as historical feedback gathered from users about already-loaded datasets. Once the user has labeled or discovered a sufficient set of columns, **LOKI** generates appropriate data loading/initialization code (e.g., a `CREATE TABLE` or Spark DataFrame initializer).

2.1 Labeling and Discovery

We assume that data arrives in unlabeled tabular format. An input table T has N columns and consists of a set of rows $t \in T$. Each row is an indexed list of attribute values $t = \langle\, t[1], \ldots, t[N] \,\rangle$ With $1 \leq A \leq N$ we write $T[A]$ to represent column A of T, or the *bag* of values located at that position in each row $T[A] = \{\!\{ \, t[A] \mid t \in T \, \}\!\}$. We define the domain of A in T, $dom_T(A)$ to be the *set* of such values (i.e., $set(T[A])$). A *schema* S is a list of N *names*, or human interpretable strings $S[1], \ldots, S[N]$ identifying each attribute. Note that this definition differs slightly from the classical definition of schemas, which here do not restrict the domain of a given attribute.

LOKI addresses two closely related problems that we term labeling and discovery. For both, we assume the existence of a utility function $u(T[A], S[A]) \in [0,1]$ that ranks the quality of a name $S[A]$ for a given column[1]. We discuss several such utility functions in this paper.

Labeling. In terms of this utility function, the goal of labeling is to take a table T and infer a schema S for it that maximizes total utility:

$$\arg\max_{S} \left(\sum_{1 \leq A \leq N} u(T[A], S[A]) \right)$$

optionally subject to the constraint that no two columns receive identical labels (i.e., for all $A \neq A'$, $S[A] \neq S[A']$).

Discovery. We refer to the dual problem as discovery: Given a name σ, we would like to know which (if any) columns are valid candidates for σ:

$$\arg\max_{A} \left(\, u(T[A], \sigma) \, \right)$$

WOODSTOCK'97, July 1997, El Paso, Texas USA
2016. ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

[1]Other factors might also be considered for the utility function (e.g., the rest of the schema or other columns of the table). We leave these considerations for future work.

optionally subject to a threshold value $0 \leq \delta \leq 1$ on the utility function. We also consider the top-k variation of this problem, where we return the k columns with highest utility.

3 OTHER CONCERNS

In addition to the challenges of labeling and discovery, there are two additional challenges in implementing **LOKI** that, at present, we solve with off the shelf techniques. First, **LOKI** need a way to infer the type (e.g., Numeric, String, or Ordinal) of columns of ingested data. For this, we adopt a simple counting-based technique [3]. We attempt to parse each value using an array of regular expressions, one for each registered type. A majority vote of successful parses in a column decides the column's type, with a threshold of 50% indicating a string column and fewer than 100 distinct values indicating an ordinal. Admittedly, more advanced techniques that relate columns, for example using functional dependencies, are possible. However majority vote suffices for most use cases, and as such we leave more interesting type annotation schemes to future work.

Second, **LOKI** needs to load data into the underlying system. Again, we use each platform's native loading scheme: `DataFrameReaders` on Spark, and native `LOAD` commands on RDBMSes, with a fall-back to manual `INSERT` operations if necessary. As before, more intricate loading options have been explored by others [1, 2], but are beyond the scope this paper.

4 KNOWLEDGE-BASE

4.1 Types of Matching Rules

Approximate.

Outline approximate matching rules: Expert heuristics, rules, etc...

- How are approximate KB entries encoded?
- How do we unify different types of heuristics?

Exact.

Outline exact matching rules: Data identity, recording/querying feedback. Merging conflicts.

- How is feedback saved in the KB.
- What happens in response to conflicting feedback.

4.2 Data Sketching for Similarity

Overview sketching data for similarity.

Data type taxonomy

- Numeric Data (Sketch = Distribution)
- Textual Data (N-Gram distribution?,)
- Enum Types (Overlap, Concept-Similarity)
- ...?

4.3 Numeric Data

Focus on the challenges of sketching numeric types

5 EXPERIMENTS

Experiments

6 RELATED WORK

Related work

7 FUTURE WORK

Future work...

- Meta-queries for columns that *could* be in a query.
- Discovery of meta-data (e.g., units)
- Automatic translation/transformation (units, structure – GPS vs Textual)

REFERENCES

[1] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. 2012. NoDB: efficient query execution on raw data files. In *SIGMOD Conference*. ACM, 241–252.
[2] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2007. Database Cracking. In *CIDR*. www.cidrdb.org, 68–78.
[3] Ying Yang, Niccolo Meneghetti, Ronny Fehling, Zhen Hua Liu, and Oliver Kennedy. 2015. Lenses: An On-Demand Approach to ETL. *Proc. VLDB Endow.* 8, 12 (2015).