Wisteria: Nurturing Scalable Data Cleaning Infrastructure

Daniel Haas, Sanjay Krishnan, Jiannan Wang, Michael J. Franklin, Eugene Wu [†*]
AMPLab, UC Berkeley [†]Columbia University
{dhaas,sanjay,jnwang,franklin}@cs.berkeley.edu, ewu@cs.columbia.edu

ABSTRACT

Analysts report spending upwards of 80% of their time on problems in data cleaning. The data cleaning process is inherently iterative, with evolving cleaning workflows that start with basic exploratory data analysis on small samples of dirty data, then refine analysis with more sophisticated/expensive cleaning operators (e.g., crowdsourcing), and finally apply the insights to a full dataset. While an analyst often knows at a logical level what operations need to be done, they often have to manage a large search space of physical operators and parameters. We present Wisteria, a system designed to support the iterative development and optimization of data cleaning workflows, especially ones that utilize the crowd. Wisteria separates logical operations from physical implementations, and driven by analyst feedback, suggests optimizations and/or replacements to the analyst's choice of physical implementation. We highlight research challenges in sampling, in-flight operator replacement, and crowdsourcing. We overview the system architecture and these techniques, then provide a demonstration designed to showcase how Wisteria can improve iterative data analysis and cleaning. The code is available at: `http://www.sampleclean.org`.

1. INTRODUCTION

The ease of acquiring and merging many large-scale data sources has led to a prevalence of dirty data. Unfortunately, blindly using results that are derived from dirty data can lead to hidden yet significant errors in modern data-driven applications, so data must be cleaned before it is used. But because data cleaning is often specific to the domain, dataset, and eventual analysis, analysts report spending upwards of 80% of their time on problems in data cleaning [9]. The analyst is faced with a breadth of possible errors that are manifest in the data and a variety of options to resolve them. She must go through the cleaning process via trial and error, deciding for each of her data sources what to extract, how to clean it, and whether that cleaning will significantly change results.

Data cleaning is inherently iterative and Figure 1 shows a common progression for the development of a data cleaning plan, in this case the extraction of a restaurant's address from its unstructured webpage. While this operation can easily be represented at a *logical* level by its input and output schema, there is a huge space of possible *physical* implementations of the logical operators. For example, extraction could depend on manually specified rules (*rule-based*), use models trained on previously extracted ground truth records (*learning-based*), ask crowd workers to extract the desired data fields (*crowd-based*), or some combination of the three (e.g., active learning, which uses crowd workers to provide labels for a learning-based approach). Even after selecting (say) a crowd-based operator, many parameters might influence the quality of the output data or the speed and cost of cleaning: the number of crowd workers who vote on the extraction for a given webpage, the amount each worker is paid, etc. A priori, a data analyst has little intuition for what physical plan will be optimal in this large space.

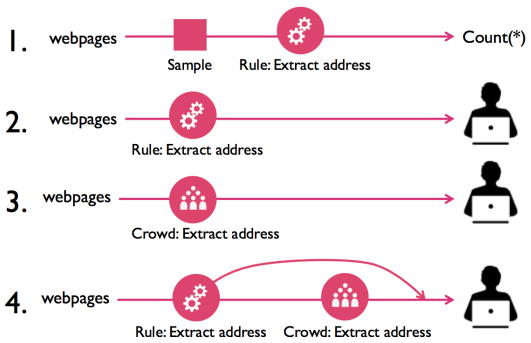

Figure 1: Example iterations on the design of the portion of a cleaning plan that extracts restaurant addresses from their unstructured webpages. 1) An exploratory plan that uses a sample to evaluate a simple address extraction method. 2) A plan that applies the method to the entire dataset. The quality is unsatisfactory. 3) An alternate plan that uses manual crowd extraction. The quality is now high, but the crowd-based extractor is slow. 4) A hybrid plan that sends only difficult webpages to the crowd, maximizing accuracy without sacrificing latency.

Note that in the evolution of the data cleaning plan in Figure 1, our data analyst needed to make many decisions manually about the choice of physical operators by reasoning about their latency, accuracy, and cost. Making the wrong decision, for example using the crowd when it only marginally improves accuracy, can be very costly. A general, scalable, and interactive system that supports rapid iteration on candidate plans would greatly aid this process.

Existing systems seldom address the end-to-end iterative data cleaning process described above. Extract-transform-load (ETL) systems [1–3] require developers to manually write data cleaning rules and execute them as long batch jobs, and constraint-driven tools allow analysts to define "data quality rules" and automatically propose corrections to maximally satisfy these rules [6]. Unfortu-

[*]Work conducted while visiting UC Berkeley

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.
Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

nately, neither provide the opportunity for iteration or user feedback, inhibiting the user's ability to rapidly prototype different data cleaning solutions. Projects such as Wrangler [4,8] and OpenRefine [15] support iteration with spreadsheet-style interfaces that enable the user to compose data cleaning sequences by directly manipulating a sample of the data and applying these sequences to the full dataset. However, they are limited to specific cleaning tasks such as simple text transformations, do not support crowd-based processing at scale, and cannot incorporate user feedback to optimize the physical implementation of the data cleaning sequences. Crowd-based [7,13] systems have been proposed to relieve the data analyst of the burden of rule specification or manual cleaning, but are usually specific to a single cleaning task (e.g., [5,7,11,12]), preventing end-to-end optimization of the entire cleaning plan. These existing limitations suggest the need for a system that is general enough to adapt to a wide range of data cleaning applications, scales to large datasets, and natively supports fast-feedback interactions to enable rapid data cleaning iteration.

In this paper, we introduce Wisteria, a system designed to support the iterative development and optimization of data cleaning plans end to end. Wisteria allows users to specify declarative data cleaning plans composed of rule-based, learning-based, or crowd-based operators, then iterate rapidly on plans with cost-aware recommendations for improving the accuracy or latency of a plan. The effects of a plan can be viewed early using sampling and approximate query processing techniques [16].

Supporting these capabilities requires a combination of careful engineering as well as tackling several research challenges:

- **Sampling**: We provide sampling as a first-class logical operator for data cleaning plans that tolerate approximation, and use it to speed up iteration on early-stage plans.
- **Recommendation**: We recommend cost-aware changes to in-flight cleaning plans that allow users to trade off accuracy and latency, and provide efficient mechanisms for implementing recommended changes without re-executing the plan on already cleaned tuples.
- **Crowd Latency**: We leverage techniques for straggler mitigation [14] and model crowd worker speed and accuracy to reduce the (often rate-limiting) latency of crowd data cleaning, consistently retrieving results in seconds rather than hours.

In our demonstration, we will run an entity resolution plan on two restaurant datasets, and show how Wisteria can be used to 1) specify, modify, and execute a data cleaning plan, 2) quickly clean a sample to characterize how a plan is performing, and 3) observe the same cleaning plans running on multiple datasets. Users can execute plans over a live crowd that uses the audience as workers, or a simulated crowd that uses pre-collected crowd responses. The dashboard (Figure 3) also provides a live inspection interface to view the status of the cleaning plan as it executes.

2. SYSTEM ARCHITECTURE

In this section, we provide a brief overview of the Wisteria system and its APIs. Figure 2 depicts the system architecture.

2.1 Architecture Overview

The Wisteria architecture provides UI, language, and systems tools for building data cleaning plans. Users interact with the system through the **Planning UI**, which allows them to compose data cleaning workflows from modular operators. These workflows are represented as expressions in our data cleaning language (section 2.2), then synthesized as data cleaning plans by our **DSL compiler**. As the **Data Cleaning Plan Executor** executes the compiled plans, users can interact with the plans via tight feedback loops in two ways. First, users can issue queries to the Sampling-based Approximate Query Processing **(SAQP) module** and observe approximate results based on the data that has been cleaned thus far. Second, the **Recommendation Engine** displays a set of suggested modifications to the active cleaning plan (for example, making a similarity join more permissive) in the **Planning UI**, and users can update the data cleaning plan in-flight by accepting a suggestion and using the **Hot Swapper** to modify components of the pipeline. Intermediate results and cleaned data are maintained in a **Lineage and Storage engine** that tracks each tuple's lineage in order to enforce the semantics of hot-swapping correctly on in-flight tuples. Logical cleaning operators may have a number of physical implementations (section 2.3). Automated rule-based or learning-based operators leverage Spark and MLLib for efficient distributed computation, and operators that require human intervention call out to Wisteria's **Crowd Manager** API, which renders and displays data cleaning tasks to crowd workers from multiple crowds (e.g., Amazon Mechanical Turk) in a web-based **Cleaning UI** for processing.

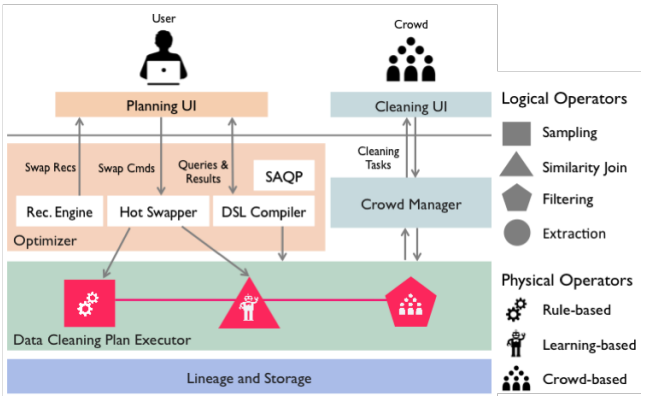

Figure 2: Wisteria system architecture, with an example entity resolution plan.

2.2 Cleaning DSL

We provide a language for specifying the composition of data cleaning operators. The logical operators define the input and output behavior of the operation and the physical operators specify the implementation. The general syntax of this language is:

```
<logical operator> on <relations>
  with <physical operators> , <params>
```

These expressions are composable. For example, the following represents the cleaning plan in Figure 2 (an entity resolution plan):

```
Filtering on (
    SimilartyJoin on (
        Sampling on BaseTable
        with Uniform)
    with Jaccard, thresh=0.8)
with CrowdDeduplication, numVotes=3
```

Additionally, Wisteria provides integration of our DSL with Scala/Apache Spark, allowing DataFrames (Spark RDDs with additional schema information) to serve as base tables in expressions.

2.3 Cleaning Operators

Wisteria supports a small set of operators that can express a wide variety of common data cleaning workflows. For example, the pipeline depicted in Figure 2 performs crowd-based entity resolution: the SimilarityJoin operator generates candidate tuple pairs

(the *blocking* step), and the crowd-based Filter operator uses humans to identify duplicates from the candidates (the *matching* step). Additional operators include Extraction and Sampling.

Individual logical operators have multiple physical implementations, each with its own cost, latency, and accuracy profile. For example, crowd-based implementations tend to be high cost, high latency, and high accuracy, whereas rule-based implementations tend to be low cost, low latency, and low accuracy. The `with` clause of our data cleaning language allows users to explicitly specify physical operators, and Wisteria's recommendation engine suggests pipeline modifications to navigate the tradeoff space.

3. RESEARCH CHALLENGES

To support evolving data quality needs, there are three main research challenges in Wisteria: (1) sampling, (2) recommendation, and (3) crowd sourcing.

3.1 Sampling

In prior work, we explored the problem of estimating aggregate query results over dirty data [10,16]. In SampleClean [16], we found that aggregate queries can often be answered with very high accuracy (i.e 99%) with only a small fraction of clean data, and we can clean just enough for the application's data quality requirements. In Wisteria, we implement sampling as a logical operator that can be used for quickly prototyping and optimizing workflows on samples of data and then transferring these optimizations to full datasets. We find that many important features of iterative data cleaning workflows can be posed as aggregate queries on samples with confidence intervals. If we want to know whether a data cleaning operation has a significant effect, we can use a query to test the effects of this operation on a sample. For example, if we are deduplicating restaurant categories, we can count the number of Chinese restaurants in a sample to test if different deduplication algorithms significantly affect the count. Sampling and result estimation are salient features of Wisteria that allow us tune parameters and provide recommendations for chaning a workflow without evaluating all possible workflows on the full data. Next, we discuss how we can efficiently generate these recommendations.

3.2 Recommendation

We also have implemented a basic recommendation engine that recommends changes to a data cleaning plan based on user feedback. A user can specify a set of ground truth tuples, and our system will optimize over data cleaning plans that best reproduce the ground truth. There are three types of recommendations: (1) parameter change, (2) operator replacement, and (3) operator addition. To realize and execute the recommendations, we use caching and lineage to efficiently re-evaluate a workflow.

Parameter Change. Many of the physical operators in Wisteria have tunable parameters, whose values are often very dataset-specific, and the user feedback gives us a way to evaluate the quality of the initial parameter choice. For example, Similarity Joins have a similarity threshold and a similarity function. Increasing this threshold reduces the selectivity of the join, and Wisteria needs to choose a threshold that maximizes accuracy. This problem can be solved by a minimum-cost spanning tree over a similarity graph (edges represent non-zero similarity) over tuples.

Operator Replacement. Wisteria recommends changes to physical operators when the user indicates that they are not satisfied with the output. For example, we can treat user feedback as a proxy for crowd labels and estimate the value of replacing a physical operator with an active learning variant. Additionally, we can try different variants of automated operators to test how accurate they are with respect to the user feedback.

Operator Addition. There are also cases where we may want to add another physical operator, while still preserving the logical input-output behavior of the workflow. It is common in extraction tasks to have most tuples accurately extracted with an automated extractor but only a small subset requiring additional inspection. For these cases, we can add a crowd-based Filter operator to separate these examples for additional cleaning.

Cost Estimates. Of course, changing plans when using crowdsourcing may significantly change its cost. For every recommendation, we estimate the number of additional tuples processed by the crowd operators and provide the user with an estimated cost. Based on a user-specified cost per task, we estimate the number of tasks needed to clean the dataset.

Caching allows for result re-use if a downstream operator is modified or added. If the system has sufficient memory, then we can naïvely cache all intermediate results. Otherwise, the key challenge is to select a subset of results to cache. We choose which results to cache by integrating the caching framework with our recommendation engine. When we make a recommendation for a change, we must cache the preceding operator.

Lineage allows us to understand how results change if upstream operators are modified. For example, decreasing a similarity join threshold increases the number of output pairs without affecting existing output pairs. The key property here is monotonicity, and some types of monotone Filter and SimilarityJoin operators are data cleaning analogs for a Select-Join relational algebra. We can therefore model upstream hot-swapping as an incremental view maintenance problem and update the final result based on the insertion or deletion of tuples earlier in the plan.

3.3 Crowdsourcing

Working with crowds is inherently challenging. Unlike when using automated operators, the accuracy and speed of processing each tuple varies widely with the crowd worker assigned to it. Completion time of an operator depends on the response times of individual workers, and on real-world crowdsourcing platforms, the distribution of response latencies is highly skewed; analogous to the straggler problem in distributed systems. We address this problem by maintaining a pool of high-speed, high-quality crowd workers and develop task routing strategies that can avoid assigning tasks to slow workers and leverage redundancy to significantly reduce the time that is required to clean data with the crowd. Additionally, active learning techniques reduce the number of tuples that require crowd work to clean the data.

4. DEMONSTRATION

In this section, we detail the proposed demonstration. The objective of this demonstration is to illustrate how Wisteria enables the rapid iterative construction of data cleaning plans and the ability to transfer workflows between similar dirty datasets.

4.1 Datasets

In the demo, we will consider cleaning workflows on three different datasets. The first dataset contains 858 Zagat reviews[1], each tagged with the cuisine of the restaurant reviewed (e.g. "Chinese" or "French"). The second dataset, which is similar to the first, is from Yelp [2] and contains 58,127 restaurant records that are also

[1] `cs.utexas.edu/users/ml/riddle/data/restaurant.tar.gz`

[2] `https://www.yelp.com/academic_dataset`

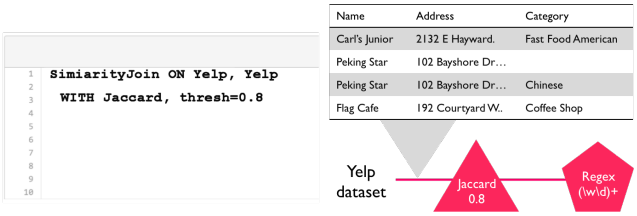

Figure 3: The dashboard contains both a visual interface and a text box to specify data cleaning operations. When the user is satisfied, she can run the plan and see the results on the right.

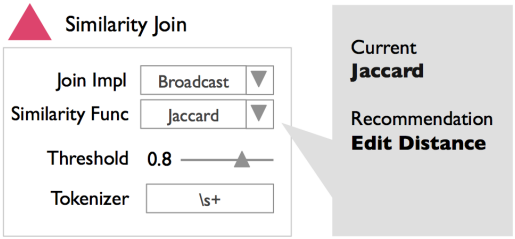

Figure 4: The operator view lists the parameters of an operator. Users can view recommended changes and modify parameters on the fly.

tagged with a category. The third dataset consists of $3,049,914$ records of liquor sales from the state of Iowa[3], including the store where the purchase occurred and the items and cost of the purchase. In all three datasets, categorical columns are inconsistent across records (e.g. cuisine tags for "Chinese" vs. "Chinese Cuisine"), records are duplicated, and formatting errors abound. We will use Wisteria to resolve these errors using Extraction and Entity Resolution, then run aggregate queries over the cleaned datasets.

4.2 Demo Walkthrough

Below, we detail the steps of the proposed demonstration. A screenshot of the dashboard interface is illustrated in Figure 3.

Step 1: Participants will select a dataset (e.g., the Zagat dataset), and load a pre-populated data cleaning plan and target query for it. Participants must first extract the columns of the dataset into the proper schema using a regex-based Extraction. Then, participants will be able to manually tune the Enity Resolution by choosing between Similarity Join implementations, adjusting the thresholds for the Similarity Join, and adding a crowdsourced filtering step.

Step 2: At all times, the interface will display a representative sample of the cleaning plan's input and output and the results of the target query so that the participant can see how cleaning affects the data. If a plan modification adds crowdsourcing, participants can complete crowd tasks in Wisteria's crowd interface.

Step 3: Participants re-evaluate and adjust their plan by clicking on an operator (Figure 4). This view will show the system's recommended changes to the operator and allow the participant to make those changes easily. For example, Figure 4 shows a recommendation to change the similarity metric from Jaccard to Edit Distance since the attribute in question does not have many tokens.

Step 4: Participants can then switch datasets. Switching between restaurant datasets (Zagat and Yelp) demonstrates reuse of the same plan on novel data, while switching to the alcohol dataset demonstrates that Wisteria is effective across data domains.

[3] `data.iowa.gov/Economy/Iowa-Liquor-Sales/m3tr-qhgy`

5. CONCLUSION

The prevalence of dirty data is a fundamental obstacle to modern data-driven applications. We introduced Wisteria, a system that supports the iterative development of data cleaning workflows. Wisteria allows the user to construct, adapt, and optimize cleaning plans with automated parameter recommendations, separating logical data cleaning operators from their physical implementations (e.g., rules, learning, or crowdsourced). In our demo, we illustrate how Wisteria can be used to clean three real datasets, Zagat, Yelp, and Iowa Liquor Sales, with different physical implementations of the same logical Extraction and Entity Resolution workflow. The physical implementations, while the same at a logical level, have different cleaning accuracies and our system aids the user in selecting the best options. We have released an initial version of the code containing the core mechanisms for specifying cleaning plans, the operator API, and implementations of several physical operators.

We thank Juan Sanchez for his help in the design and implemention of this system. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE 1106400. This research is also supported in part by NSF CISE Expeditions Award CCF-1139158, LBNL Award 7076018, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, The Thomas and Stacey Siebel Foundation, Adatao, Adobe, Apple, Inc., Blue Goji, Bosch, C3Energy, Cisco, Cray, Cloudera, EMC2, Ericsson, Facebook, Guavus, HP, Huawei, Informatica, Intel, Microsoft, NetApp, Pivotal, Samsung, Schlumberger, Splunk, Virdata and VMware.

6. REFERENCES

[1] Apache falcon. `http://falcon.apache.org`.
[2] Informatica. `https://www.informatica.com`.
[3] Talend. `https://www.talend.com/solutions/etl-analytics`.
[4] Trifacta. `http://www.trifacta.com`.
[5] Z. Chen and M. Cafarella. Integrating spreadsheet data via accurate and low-effort extraction. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1126–1135. ACM, 2014.
[6] M. Dallachiesa, A. Ebaid, A. Eldawy, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. Nadeef: a commodity data cleaning system. In *SIGMOD Conference*, pages 541–552, 2013.
[7] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.
[8] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.
[9] S. Kandel, A. Paepcke, J. Hellerstein, and H. Jeffrey. Enterprise data analysis and visualization: An interview study. *VAST*, 2012.
[10] S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, and T. Kraska. Stale view cleaning: Getting fresh answers from stale materialized views. *Proc. VLDB*, 8(12), 2015.
[11] C. Mayfield, J. Neville, and S. Prabhakar. Eracer: a database approach for statistical inference and data cleaning. In *SIGMOD*, 2010.
[12] H. Park and J. Widom. Crowdfill: Collecting structured data from the crowd. In *SIGMOD*, 2014.
[13] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *CIDR*, 2013.
[14] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica. The power of choice in data-aware cluster scheduling. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, pages 301–316. USENIX Association, 2014.
[15] R. Verborgh and M. De Wilde. *Using OpenRefine*. Packt Publishing Ltd, 2013.
[16] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD Conference*, pages 469–480, 2014.