Bulletin of the Technical Committee on

Data Engineering

June 2018 Vol. 41 No. 2

IEEE Computer Society

Letters

Letter from the Editor-in-Chief . . . . . *David Lomet* 1
Letter from the Special Issue Editor . . . . . *Guoliang Li* 2

Special Issue on Large-scale Data Integration

Data Integration: The Current Status and the Way Forward . . . . . *Michael Stonebraker, Ihab F. Ilyas* 3
BigGorilla: An Open-Source Ecosystem for Data Preparation and Integration . . . . . *Chen Chen, Behzad Golshan, Alon Halevy, Wang-Chiew Tan, AnHai Doan* 10
Self-Service Data Preparation: Research to Practice . . . . . *Joseph M. Hellerstein, Jeffrey Heer, Sean Kandel* 23
Toward a System Building Agenda for Data Integration (and Data Science) . . . . . *AnHai Doan, Pradap Konda, Paul Suganthan G.C., Adel Ardalan, Jeffrey R. Ballard, Sanjib Das,Yash Govind, Han Li, Philip Martinkus, Sidharth Mudgal, Erik Paulson, Haojun Zhang* 35
Explaining Data Integration . . . . . *Xiaolan Wang, Laura Haas, Alexandra Meliou* 47
Making Open Data Transparent: Data Discovery on Open Data . . . . . *Renée J. Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q. Pu, Periklis Andritsos* 59
Big Data Integration for Product Specifications . . . . . *Luciano Barbosa, Valter Crescenzi, Xin Luna Dong, Paolo Merialdo, Federico Piai, Disheng Qiu, Yanyan Shen, Divesh Srivastava* 71
Integrated Querying of SQL database data and S3 data in Amazon Redshift . . . . . *Mengchu Cai, Martin Grund, Anurag Gupta, Fabian Nagel, Ippokratis Pandis, Yannis Papakonstantinou, Michalis Petropoulos* 82
Robust Entity Resolution Using a CrowdOracle . . . . . *Donatella Firmani, Sainyam Galhotra, Barna Saha, Divesh Srivastava* 91
Human-in-the-loop Rule Learning for Data Integration . . . . . *Ju Fan, Guoliang Li* 104

Conference and Journal Notices

ICDE 2019 Conference 116
TCDE Membership Form . . . . . back cover

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

About the Bulletin

This June's issue is the start of the editorial tenure of four new editors. I am both proud and humbled by my continued success in recruiting top notch technical people when I invite them to be Bulletin editors. This success is, indeed, the "secret sauce" responsible for the Bulletin's continued success.

With the above as prelude, I am proud to announce that the new Bulletin editors are, in alphabetical order, Philippe Bonnet of ITU Copenhagen, Joseph Gonzalez of UC Berkeley, Guoliang Li of Tsinghua University, and Alexandra Meliou of UMASS Amherst. Their contact information appears on the inside cover of this issue. This talented team will be bringing you Bulletin issues over the next two years.

The Current Issue

There is a bias among some in the research community to focus on solvable problems. You create a program or system and it either solves the problem or it does not. Meanwhile, in the real user community, technical people struggle at tasks that have to be done where it is either impractically difficult or impossible to provide a technical approach that simply solves "the problem".

Data integration is such a problem. It is important because there are 10's, 100's, even 1000's of data sources that all too frequently need to be "integrated" in some way to solve a real world problem. This problem is sufficiently hard and costly that incremental improvements can be very important. For example, reducing the amount of human involvement in the data integration activity can produce important cost savings and even lead to better results.

Data integration is the subject of the current issue. A quick look at the table of contents will show you that data integration has become a "first class" problem for our field. Authors come from universities, large companies, former start-ups, etc., with the divisions between these homes in flux. This illustrates the excitement in the area. Guoliang Li has reached out to these communities in his effort to give you this survey of the field. The issue is a snapshot in time of this challenging area, where the state of the art is improving continually. I want to thank Guoliang for a very successful issue on an important topic.

ICDE 2019

The current issue includes an ICDE 2019 call for papers on page 116. ICDE is a premier database conference and the flagship conference of the Technical Committee on Data Engineering. This year's submission deadlines are earlier than in the past, and there is a two round submission process. The intent is to provide authors earlier feedback and a better opportunity to publish their work earlier in the year. ICDE 2019 is in Macau, which has, via adjacency to Hong Kong, great air connections to the rest of the world. I hope to see you there.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

Data integration is a long-standing problem over the past few decades. As more than 80% time of a data science project is spent on data integration, it becomes an indispensable part in data analysis. In recent few years, tremendous progress on data integration has been made from systems to algorithms. In this issue, we review the challenges of data integration, survey data integration systems (e.g., Tamr, BigGorilla, Trifacta, PyData), report recent progresses (e.g., explaining data integration, data discovery on open data, big data integration pipeline for product specifications, integrated querying of table data and S3 data, crowd-based entity resolution and human-in-the-loop rule learning), and discuss the future of this filed.

The first paper, "Data Integration: The Current Status and the Way Forward" by Michael Stonebraker and Ihab F. Ilyas, discusses scalable data integration challenges based on the experience at Tamr, and highlights future directions on building end-to-end scalable data integration systems, such as data discovery, human involvement, data exploration, and model reusability.

The second paper, "BigGorilla: An Open-Source Ecosystem for Data Preparation and Integration" by Alon Halevy et al, presents BIGGORILLA, an open-source resource for data preparation and integration. The paper describes four packages – an information extraction tool, a schema matching tool, and two entity matching tools.

The third paper, "Self-Service Data Preparation: Research to Practice" by Joseph M. Hellerstein et al, reviews self-service data preparation, which aims to enable the people who know the data best to prepare it. The paper discusses the key tasks in this problem and reviews the Trifacta system on how to handle these tasks.

The fourth paper, "Toward a System Building Agenda for Data Integration (and Data Science)" by Anhai Doan et al, advocates to build data integration systems by extending the PyData system and developing more Python packages to solve data integration problems. The paper provides an integrated agenda of research, system building, education, and outreach. The paper also describes ongoing work at Wisconsin.

The fifth paper, "Explaining Data Integration", by Laura Haas et al, reviews existing data integration systems with respect to their ability to derive explanations. The paper presents a new classification of data integration systems by their *explainability* and discusses the characteristics of systems within these classes. The authors also present a vision of the desired properties of future data integration systems with respect to explanations.

The sixth paper, "Making Open Data Transparent: Data Discovery on Open Data" by Renée J. Miller et al, discusses the problem of data discovery on open data, e.g., open government data. The paper considers three important data discovery problems: finding joinable tables, finding unionable tables, and creating an organization over a massive collection of tables.

The seventh paper, "Big Data Integration for Product Specifications" by Divesh Srivastava et al, presents an end-to-end big data integration pipeline for product specifications. The paper decomposes the problem into different tasks from source and data discovery, to extraction, data linkage, schema alignment and data fusion.

The eighth paper, "Integrated Querying of SQL database data and S3 data in Amazon Redshift" by Yannis Papakonstantinou et al, discusses query planning and processing aspects in Redshift, which provides integrated, in-place access to relational tables and S3 objects. The paper proposes techniques to optimize Redshift.

The ninth paper, "Robust Entity Resolution Using a CrowdOracle" by Divesh Srivastava et al, studies the problem of crowdsourcing-based entity resolution. The paper summarizes the CrowdOracle pipelines and describes a common framework consisting of simple operations.

The last paper, "Human-in-the-loop Rule Learning for Data Integration" by Ju Fan and Guoliang Li, proposes human-in-the-loop rule learning for effective data integration. The approach first generates a set of candidate rules, proposes a machine-based method to learn a confidence for each rule using generative adversarial networks, and devises a game-based crowdsourcing framework to refine the rules.

I would like to thank all the authors for their insightful contributions. I hope you enjoy reading the papers.

Guoliang Li
Tsinghua University
Beijing, China

Data Integration: The Current Status and the Way Forward

Michael Stonebraker
MIT
stonebraker@csail.mit.edu

Ihab F. Ilyas
University of Waterloo
ilyas@cs.uwaterloo.ca

Abstract

We discuss scalable data integration challenges in the enterprise inspired by our experience at Tamr[1]. *We use multiple real customer examples to highlight the technical difficulties around building a deployable and usable data integration software that tackles the data silos problem. We also highlight the practical aspects involved in using machine learning to enable automating manual or rule-based processes for data integration tasks, such as schema mapping, classification, and deduplication.*

1 Introduction

In this paper, we comment on the status and the issues in data integration from the perspective of Tamr, a commercial company that provides a novel solution to a long-standing challenge, namely, traditional *enterprise data integration*. Rather than relying on a rule-based approach, Tamr employs supervised machine learning as the primary way of combining large numbers of data sources. The company is based on the Data Tamer academic system [13].

Most large businesses are decomposed into independent business units to facilitate agility. Such units are typically free to "do their own thing", without being hindered by corporate-wide issues. For example, adopting a specific global schema for a specific entity type (e.g., customers) across all units is often impossible as the needs of these units are different. Waiting for consensus across all the business units means it would take forever to get anything done. This leads to *data silos* (one for each unit), where similar data are stored with different granularity, schema, and even contradicting and inconsistent details. A typical enterprise has many such silos, and a major goal of many enterprises is after-the-fact integration of silos for business gain. Such business gain often involves cross selling, a better understanding of the customer base or lowering the expenses of product lines. Such activities span many business units and require classifying, integrating, linking and aggregating the data from their silos.

Example 1 (GE Supplier Integration): GE has approximately 75 active procurement systems, with approximately two million records. The CFO determined that the company could save $\$100M$ per year from the following strategy: when the contract with an external vendor comes up for renewal in one of these procurement systems, empower the purchasing officer to discover the terms and conditions negotiated by his counterparts in other business units, and then allow him to demand most favored nation status. To accomplish this, 75 independently constructed supplier databases must be integrated. In fact, the value is mostly in the long tail since the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] www.tamr.com

major suppliers are routinely scrutinized. Hence, a strategy that integrates only the major suppliers will fail to provide additional value.

A typical data integration workflow of the 75 different supplier databases is as follows: **(a)** Raw source data is "ingested" into a common system (often a data lake); **(b)** Records are transformed into common units (say Euros to Dollars) via a series of transformation rules, modules and services; **(c)** Errors are cleaned [8]; typically 15% of the values are missing or incorrect, and most integration tools expect these errors to be at least spotted or nulls to be identified; **(d)** The different schemas of the sources are matched to line up the columns. This step is crucial in enabling comparing records across sources; **(e)** Deduplication and record linkage tools [4, 7, 10] are used to cluster records. Each cluster of records is thought to represent the same entity; and finally **(f)** Golden canonical values must be selected for columns in these clusters (e.g., a common address must be selected for each supplier) to produce a final integrated data set that can be consumed by downstream spend analytics.

In effect, 75 supplier data sets are pushed through this pipeline to generate an integrated composite data set, through which purchasing officers can find the most favorable terms and conditions, among other possible queries. The pipeline has to be able to support incremental processing of new suppliers (or deleted suppliers), without the need to redo the whole workflow from scratch on all of the source data.

The previous example focused on a schema matching and record deduplication workflow of data that represents the same real-world entity scattered across silos. However, data integration often involves other tasks, such as classifying input records to known ontologies (e.g., standards or master dictionaries), as we show in Example 2. This classification step is often interleaved with the deduplication and schema mapping steps to carry out an end-to-end integration project.

Example 2 (GE Parts Classification): In addition to the schema mapping and deduplication tasks in Example 1, a second GE problem is to classify $20M$ *spend* transactions into an existing GE classification hierarchy (a given taxonomy for various parts). A spend category could be *machine parts* and a sub category could be *bolts*. A further classification could be *stainless steel bolts*. GE engineers wrote more than 500 rules to specify which bucket a spend record was in (e.g. *taxi rides* should be classified as *travel*).

A functional interactive workflow has to support a "rollback" functionality to support false starts, or trying different configurations for each of these steps or components. The workflow must be also able to call external modules, such as web services, locally or externally developed tools, or other software packages (i.e. acting as a master to other workflows). In addition, an integration workflow has to fit into workflows run by other more general pipelines (e.g., business intelligence or analytics workflows).

In our opinion, the most interesting data integration problems are ones that require scalability. Any methodology (including manual integration) is probably capable of integrating 1000 records from each of 10 data sources. However, for problems that deal with larger data scale like the one in the previous example, scalable execution of these pipelines is a must, and in fact, has been the main hurdle in deploying data integration software in real scenarios, despite the long history of the problem in both academia and industry.

This paper discusses the current state of the art and future challenges in *scalable data integration* from the perspective of Tamr, and is informed by interacting with large enterprises over the last five years. In Section 2, we discuss why traditional solutions do not scale, the use of machine learning for automation and for replacing/augmenting rule-based approaches, and we discuss why building such systems is hard due to scale, dirtiness of data, and resistance to adoption inside the enterprise. In Section 3, we shed some light on future directions and missing pieces that need to be addressed to build scalable end-to-end data discovery and integration systems. We conclude in Section 4 by a summary and final remarks.

2 Scalable Data Integration

2.1 Traditional Solutions Do Not Scale

The traditional wisdom in dealing with large scale data integration projects is to use what is referred to as a *Master Data Management* (MDM) system, such as those available from Informatica and IBM. Using an MDM system, one performs merge/purge (deduplication and golden value selection) often using a rule-based system, which allows users and experts to declare rules for transforming and classifying input data records. Before this step, a schema mapping exercise is performed typically by drawing lines between matching columns using a GUI in a (semi) manual way.

It is not hard to see that traditional MDM systems do not scale well: manual schema mapping obviously does not scale beyond tens of sources. For example, in one `Tamr` application there are more than 1000 data sets, each with approximately 500 columns. Manually linking $500,000$ columns is clearly not workable, and a scalable and more automated solution is required.

A lesser known fact is that rule systems for classification and record linkage do not scale either, as the system grows rapidly with a large number of correlated, overlapping and sometimes contradicting rules to cover all edge cases, which presents serious maintainability issues. In Example 2, the 500 rules designed by the `GE` engineers is about the limit of what a human can understand. However, these rules only classify $2M$ (10%) of the transactions. It is impossible to imagine a rule system with $5000+$ rules, which would be required to classify all $20M$ transactions.

As a result, MDM technology simply does not scale, and should only be used on small problems that are guaranteed not to require scalability in the future. Unfortunately, non scalability is very difficult to guarantee, since large enterprises reorganize regularly and buy and sell business units like monopoly properties.

2.2 Machine Learning for Scalable Data Integration

An obvious approach to scalability is to utilize machine learning (ML) to automate the various steps, such as the ones mentioned in the previous examples. In Example 2 (parts classification), we can use the $2M$ classified records as training data for an ML classification model, which will then classify the remaining $18M$ records, and this is exactly the `Tamr` solution implemented by `GE`. We see no other alternative to achieve the required scalability. In other words, we need to automate classification decisions (classifying records, matching columns, or linking similar records) using ML models that can leverage training data, which is either collected directly from humans in the loop, or in a weak-supervision fashion via leveraging existing rule systems. There are, of course, many engineering challenges to applying machine learning at scale, and the rest of this paper will consider some of them. However, in this section, we discuss the ML tactics that are likely to be successful in the commercial marketplace.

Conventional ML models, such as regression, decision trees, random forests, SVM, and Naïve Bayes models, are well understood, and often require elaborate feature engineering exercise to provision. In the last few years, there has been dramatic interest in "deep learning" (neural networks). The pioneering work came from Google, which successfully applied neural networks to find images of cats, and this technique has been successfully applied in a variety of other image understanding and text-oriented problems. In `Tamr` applications, there is little-to-no image and text data, rather, it is essentially all structured data such as found in relational DBMSs.

As we discuss below, using deep learning in enterprise data integration applications remains challenged by the scarcity of training data, since most of these models require large quantities of labeled data to learn the classification task; and by the lack of reasonable explanations of the output decisions.

- *Training Data:* In general, generating training data in enterprise data integration tasks is a huge problem: consider three plausible terms; *IBM-SA*, *IBM Inc* and *IBM*. Intuitively, these might represent the Spanish

subsidiary, the US subsidiary and the overall company. Deciding if these should be consolidated as duplicates or kept as separate meaningful related entities can be performed by a domain expert. Even then, the decision to consolidate or not could be determined by the question being asked. However, automatically "learning" these complex relationships among these entities, and the right merge/separate decision will probably require massive number of labeled data in a deep neural network model.

In the hundreds of applications we have seen through Tamr, domain expert time is a scarce commodity that must be ruthlessly economized and cannot be simply used to label millions of training examples. These experts are typically well-paid, busy business experts, who view generating training data as a low priority task. Most of Tamr customers have a specific budget for domain expert time, and it is the "long pole in the tent" for the project. Hence, it is easy to see why ML models with less training data requirements are highly preferable than those with large training data demands (e.g., deep learning models). Advances in automating training data collection, such as the Snorkel project [11], might relax this constraint. However, in Tamr applications, we have not seen the type of rule coverage and the significant overlap of rules that Snorkel requires to produce good results. Until we effectively solve the problem of generating large and high-coverage training data, enterprise solutions will likely depend more on conventional ML classifiers with modest training requirement.

- *Explainability:* In many enterprise integration problems, one must be able to explain why the ML model took a particular action. For example, a predictive model that generates approvals for real estate loans must be explainable: *you were denied a loan because of such and such reasons*. If this explanation is not forthcoming, then lawsuits are inevitable, and adoption of these models is highly unlikely. Conventional ML is at least marginally explainable; deep learning models are not. Again, this will be an impediment to the applicability of deep learning in enterprise integration applications. It is conceivable that deep learning models will become more explainable in the future, and there is considerable research in this area. However, until this happens, we do not foresee the adoption of these models at a wide scale, or as a primary classification method in enterprise data integration tasks.

For these reasons, at least in the short-term, these conventional ML models with modest training data requirements will prevail when integrating structured data at scale.

2.3 Scalable Data Integration is Fundamentally Difficult

Consider a second example, that of Carnival Cruise Lines. It is a holding company with 10 operating brands (e.g., *Princess*, *Carnival*, and *Costa*). Each has its own mechanism of identifying spare parts and has a parts classification hierarchy. Each has its own mechanism for depoting spare parts (e.g., on the boat, on the dock, or in the warehouse). Carnival wishes to share spare parts across brands (since everybody uses the same straws, for example) and to optimize spare part placement. This requires performing parts integration at scale ($4M$ total parts).

There are two reasonable ways to proceed. First, one can directly integrate the $4M$ source parts records. Another alternative is to integrate the various classification schemes into a single master classifier, then classify the $4M$ parts, and finally, deduplicate each bucket in the master scheme. It is not obvious which way will work better, and the best choice is clearly data dependent. We do not expect such issues to be tackled by unskilled personnel anytime soon. In the meantime, engineering of data integration pipelines is not for the faint of heart; Tamr has a substantial collection of trained field engineers who help construct customer pipelines.

In addition to the large number of design choices, many of the needed algorithms have at least a quadratic complexity. For example, clustering of similar source records for deduplication has an $O(N^2)$ computation step to compare all pairs of records. For $10M$ source records, that entails close to 10^{14} comparisons. If each one requires 10 microseconds, the complete calculation will consume more than three years. This is obviously

a non-starter. A serious product that includes deduplication must knock down the complexity by blocking and parallelism [2]. Products with an $O(N^2)$ algorithms, even on custom hardware, are likely to take an unacceptable amount of time at scale.

2.4 Data Cleaning in Integration Workflows

As noted above, it is reasonable to assume that 15% of all data is missing or wrong in a typical enterprise repository. The best first steps are to look for missing values and outliers using standard techniques, and perform data wrangling transformations [9]. Specifying a collection of data cleaning rules will also bear fruit. However, to do better, we often need application-specific cleaning tools, since "one size does not fit all". There is promising work in this area, such as the `HoloClean` project [12], which uses statistical inference to impute missing values and to suggest repairs for erroneous cells. However, it is not clear how to integrate these tools in a scalable way in workflows like the ones provisioned by `Tamr`, and in which order these cleaning tools should be used when interleaved with the main integration components [1].

A promising data cleaning tactic is to generate clusters of records, thought to represent the same entity using a deduplication tool. For many attributes, only one value is appropriate, and one can use a golden record system to pick it. For example, in the `GE` supplier integration task, a single supplier must have the same Dow Jones identifier or maybe the same address. If various records contain non-duplicate values for these fields, then a golden record system can be an effective data cleaning tool.

However, to use it, one must perform deduplication to create clusters, which is fundamentally inaccurate and requires some human involvement as noted above. The subsequent golden record system also requires human input and is operating on, perhaps, inaccurate data. In effect, a human can improve the training data, improve the clustering of records or work on the golden record system. These tactics are intertwined, and it is complex to decide what to do next. Since the golden record system is a cleaning tool, these tactics must also be compared against other cleaning tactics in any end-to-end model.

In summary, the various data cleaning tactics are not independent, complicating a general model to optimize which tool to use next. A deeper understanding and explicitly modeling of this interaction among cleaning, integration, and benefit of human involvement is essential in devising a self-configurable workflow.

2.5 Human Involvement

In `Tamr`, there are two kinds of user roles; there is a workflow designer and/or manager who is typically an IT professional who designs and manages the execution of a workflow. In addition, there are domain experts, who can answer questions, sometimes via basic channels such as e-mail. Such questions include assisting validating training data, validating clusters and validating data cleaning and golden record decisions. The two roles are carefully separated. In our opinion, one cannot have a single interface for both kinds of roles. Any system with a single interface is not going to work well.

2.6 Antibodies

Inevitably a data integration project changes the power dynamics inside an enterprise. In engineering, there are often custom one-off systems that are rendered obsolete by a successful integration effort using `Tamr` software. Such engineering groups will often throw up "engineering antibodies" to try to kill off the new effort. In addition, there are often business groups that are affected positively or negatively by an integration project. The negatively affected groups often throw up business antibodies.

`Tamr` projects are invariably instituted by business people who own a particular business problem. Invariably, it is up to them to deal with both kinds of antibodies. In summary, most data integration projects at scale have a political component, which must be dealt with.

3 The Future of Data Integration

In this section, we highlight future directions and challenges that are likely to have a big impact in building end-to-end scalable data integration in the large enterprise settings.

High Provisioning and Tuning Cost: Right now, enterprise data integration is a high touch environment, requiring highly trained personnel to be involved in authoring the right workflows, engineering the right configurations in each component, choosing the most effective cleaning tactics, and overseeing the availability of sufficient and meaningful training data. Over time, we expect the problems indicated in the previous section to be rendered easier by better tools. As such, we expect this research area to be ripe for future innovation.

Data Discovery: In addition, we expect a new application area to be commercialized that of supporting the work of data scientists. For example, `Merck` has approximately 4000 Oracle databases, a large data lake, uncountable individual files and the company is interested in public data from the web. `Merck` data scientists spend at least 90% of their time finding data sets relevant to their task at hand and then performing data integration on the result. Roughly speaking, data scientists have a discovery problem to identify data sets of interest, followed by the same data integration challenge faced by traditional enterprises. No data scientist we have talked to reports spending less than 80% of his time on discovery and integration. It is an obvious challenge to lower this percentage.

Integrating data discovery activities into traditional data integration operations is a challenging task. For example, there are often many current data catalogs, which collect metadata about enterprise objects, with rudimentary metadata such as object names, and attributes types. To become interesting discovery products, such systems must include relationships between objects (e.g., *Column A in Table 1 is related to and generalizes Column B in Table 2*). In this way, if a data scientist wants to know about data sets that deal with the effect of the drug *ritalin* on mice, he will likely find several data sets dealing with the topic. In order to proceed, this user will need to know relationships between the various data sets. This will require a much more elaborate catalog system, and a step in this direction is indicated in the `Aurum` project [5, 6], where such relationships are automatically discovered using syntactic and semantic features of the various tables and columns.

Moreover, there are often several possible ways to connect data sets of interest. For example, in the MIT data warehouse, there are several ways to connect students to departments, for example, *majoring in*, *took a course from*, or *interested in*. Unlike traditional enterprise integration, where there is usually a well-defined collection of data sets to be integrated in a well-defined way, in data science, neither is well-defined. Hence, there is a join path discovery problem which must be solved. Some of these challenges are handled in the `Data Civilizer` project [3] that builds on the relationships discovered via `Aurum` to devise possible and semantically coherent join paths, which join relevant raw data sources together to feed the analytics of a specific task.

Human Involvement: A data integration project is often run by a single data scientist. The same data scientist must be both the domain expert and the project manager. To support this application, data integration systems have to be much easier to use and require much less human interaction. These are topics for future research.

Batch vs. Exploratory Workflows: A typical enterprise data integration project starts with a collection of data sets, say the 75 supplier data bases in Example 1, and the project manager pushes these data sets through a well-defined pipeline, with a well-defined goal. In contrast, a data scientist is typically in exploration mode. He tries various things, sees what works and then uses this information to explore other related tasks. We expect supporting such users will require more serious support for provenance, rollback and conditional workflows.

Model Reusability: We fully expect other uses for data integration will occur over time. Specifically, it is clear that every large enterprise on the planet has a supplier integration problem. Moreover, there is substantial commonality between the applications of enterprises. After all, everybody buys office supplies from a small number of large vendors. Hence, imagine performing data integration for a certain project, then, use this result as training data for a second project and so forth. Hence, transfer learning and the reuse of machine learning models holds enormous promise. The impediment is that enterprises tend to view their supply chains as

highly confidential. How to get the benefit of temporal machine learning without violating confidentiality is an interesting research question.

4 Conclusion

Integrating the large structured data repositories in the enterprise, which are often scattered across many silos, requires building end-to-end scalable workflows. Current semi-manual and rule-based systems simply do not scale and cannot accommodate the continuously growing data across the various business units. While machine learning techniques is an obvious way to go, multiple practical considerations arise, such as, the scarcity of training data, the need to explain the results to business owners, and the high cost of involving domain experts. These factors play a big role in choosing the right machine learning models to deploy, and in exploring the large number of design choices in provisioning data integration workflows. Moreover, data discovery and exploration tools will become essential to help data scientists find relevant data sets, and navigate through the messy data lakes that are currently built by the large enterprises.

References

[1] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.

[2] X. Chu, I. F. Ilyas, and P. Koutris. Distributed data deduplication. *PVLDB*, 9(11):864–875, 2016.

[3] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017.

[4] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[5] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *34th IEEE International Conference on Data Engineering, ICDE, Paris, France*, 2018.

[6] R. C. Fernandez, E. Mansour, A. Qahtan, A. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *34th IEEE International Conference on Data Engineering, ICDE, Paris, France*, 2018.

[7] L. Getoor and A. Machanavajjhala. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment*, 5(12):2018–2019, 2012.

[8] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4):281–393, 2015.

[9] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, pages 3363–3372, 2011.

[10] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. 2010.

[11] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, 2017.

[12] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

[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, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

BigGorilla: An Open-Source Ecosystem for Data Preparation and Integration

Chen Chen* Behzad Golshan* Alon Halevy* Wang-Chiew Tan* AnHai Doan†
*Megagon Labs †Univ. of Wisconsin
{chen,behzad,alon,wangchiew}@megagon.ai anhai@cs.wisc.edu

Abstract

We present BIGGORILLA*, an open-source resource for data scientists who need data preparation and integration tools, and the vision underlying the project. We then describe four packages that we contributed to* BIGGORILLA*:* KOKO *(an information extraction tool),* FLEXMATCHER *(a schema matching tool),* MAGELLAN *and* DEEPMATCHER *(two entity matching tools). We hope that as more software packages are added to* BIGGORILLA*, it will become a one-stop resource for both researchers and industry practitioners, and will enable our community to advance the state of the art at a faster pace.*

1 Introduction

Data preparation and data integration are long standing problems faced by industry and academia and considerable research has been carried out on different facets of these problems (e.g., [11, 12, 18, 26]). Yet, if a practitioner wishes to get her hands dirty, it is not obvious where to search for the appropriate tools that will suit her needs, or if they even exist. In contrast, the machine learning community has a "go-to" place, Scikit Learn [30], that provides tools for the practitioners working in Python. The lack of such a resource for data integration and preparation is a barrier to the dissemination of the technology developed by our community and limits its impact. Part of the reason that such a resource does not exist is that the space of data preparation and integration is large and is closely intertwined with other data management tools. For example, a typical data integration scenario might require several steps, such as data acquisition, extraction, cleaning, schema matching and mapping, entity matching, and workflow management. Moreover, today several of these steps also have to deal with data at scale. Furthermore, depending on the application, these steps often need to be adapted and repeated in different ways to suit the needs of the application's domain.

Motivated by the desire to create a one-stop resource for data scientists working on data preparation and integration, we initiated BIGGORILLA[1] and seeded it with a few high-quality packages. BIGGORILLA is an open-source ecosystem for data preparation and integration and is currently supported in Python, which is the most popular programming language used by data scientists and taught widely. The over-arching goal of BIGGORILLA is to become the go-to resource for data preparation and integration. In turn this means the website of BIGGORILLA should contain pertinent information for data scientists and pointers to relevant and

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1]https://www.biggorilla.org. Data preparation and integration is a big, hairy, nasty problem and hence the name BIGGORILLA. The problem has also been referred to as the "800 pound gorilla of Big Data".

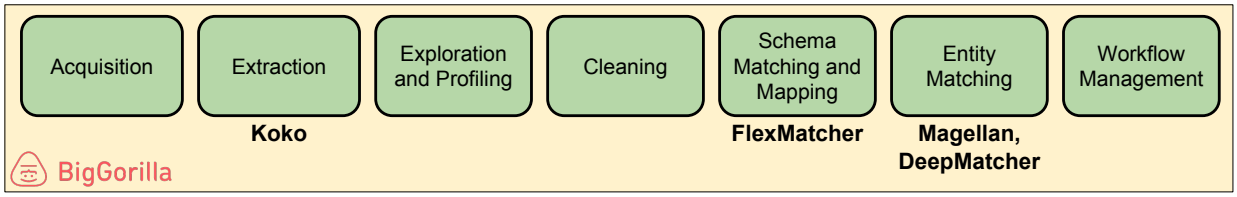


Figure 1: BIGGORILLA categorizes software into different components that are typical in a data preparation and data integration pipeline. The software packages that we contributed (KOKO, FLEXMATCHER, MAGELLAN, DEEPMATCHER) are listed under their respective components. The BIGGORILLA community will benefit as packages for more of these components get contributed.

freely available resources. By being open-source, BIGGORILLA also hopes to facilitate an active community and naturally surface a set of actively maintained software around data preparation and data integration.

BIGGORILLA cannot succeed without the contributions of the community at large. We hope to harness the knowledge of the community to add to and improve the content of the website over time. To begin this effort, we have listed on the website a number of popular software packages in each component. We hope the community will add to these lists, and in cases where they become long, will help us rank the packages by their quality and appropriateness for different contexts. In some cases, we plan to enlist a set of experts to curate a smaller set of software to place on top of the list.

Figure 1 shows the components that are often used in a typical data preparation and integration pipeline. While there are already several open-source packages in each component, the community will benefit from the availability of more packages particularly in data acquisition, cleaning, and workflow management.

The BIGGORILLA website also features a number of tutorials/examples that showcase how some of the software packages may be used. This is another way the community can contribute – we welcome example tasks that have been written which use BIGGORILLA components. Once a repository of examples is in place, it becomes easier for others to learn, through the examples, about data preparation and integration and how to use some of the software packages.

In the rest of this paper we describe four software packages that we contributed to BIGGORILLA: KOKO [38], an information extraction package, FLEXMATCHER [14] for schema matching, and MAGELLAN [17] and DEEPMATCHER for entity matching. We also describe the application of these packages to projects at several companies (including Recruit Holdings) and UW-Madison.

2 Koko: A System for Scalable Semantic Querying of Text

KOKO [38] is an information extraction system for extracting tuples from text. It supports declarative specification of conditions on the surface syntax of sentences and on the structure of the dependency parse trees of sentences. While this idea has been explored in the past [35, 36], KOKO also supports conditions that are forgiving to linguistic variations of expressing concepts and allows to aggregate supporting evidence from the entire document in order to filter extractions. Furthermore, KOKO scales to millions of documents by exploiting a multi-indexing scheme and heuristics for efficient extractions.

In what follows, we present KOKO with some examples. The interested reader can find further details in [38].

Surface syntax and conditions over dependency parse trees Suppose we wish to extract foods that were mentioned as "*delicious*" in text. We could try to specify an extraction pattern that looks for the word "*delicious*" preceding a noun that is known to be in the category of foods. However, the preceding pattern would not work for the sentence "*I ate a delicious and salty pie with peanuts*" since the word "*delicious*" does not immediately precede the word "*pie*", and it also precedes the word "*peanuts*" which were not deemed delicious. Some sentences are

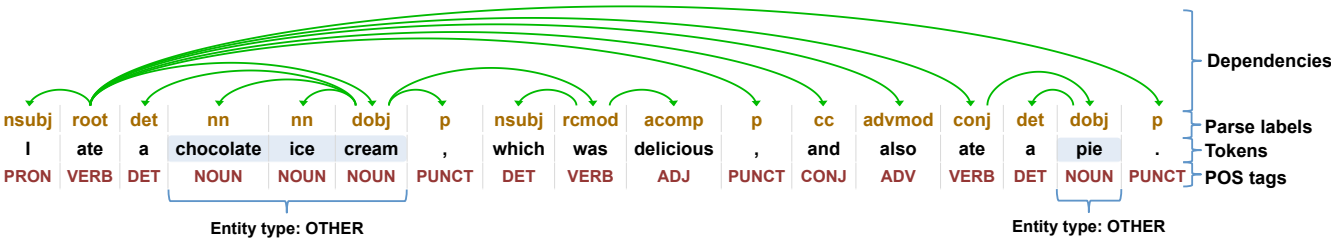


Figure 2: A sentence and its dependency tree annotated with parse labels [25], tokens, POS (Part-Of-Speech) tags [32], and entity types. This dependency tree is generated from Google Cloud NL API [19].

```
extract e:Entity, d:Str from input.txt if
(/ROOT:{
        a = //verb,
        b = a/dobj,
        c = b//"delicious",
        d = (b.subtree)
} (b) in (e))
```

Figure 3: Surface syntax and conditions over dependency parse trees.

```
extract x:Entity from "input.txt" if ()
satisfying x
        (str(x) contains "Cafe" {1}) or
        (str(x) contains "Roasters" {1}) or
        (x ", a cafe" {1}) or
        (x [["serves coffee"]] {0.5}) or
        (x [["employs baristas"]] {0.5})
with threshold 0.8
excluding (str(x) matches "[Ll]a Marzocco")
```

Figure 4: Similarity and aggregation conditions over the entire input.txt

even more nuanced. For example, "*I ate a chocolate ice cream, which was delicious, and also ate a pie*" the word "*delicious*" comes *after* "*ice cream*" which makes the extraction even more challenging.

The first example KOKO query (shown in Figure 3) can correctly extract "*ice cream*" and does not extract "*pie*". More precisely, it extracts pairs of entity e and string d from sentences (defined in the extract clause), where e is an entity that is described as "*delicious*" and the string d is the description of the entity where the word "*delicious*" occurs. The query operates simultaneously over the surface syntax and dependency parse tree of each input sentence in "input.txt". An example of a dependency parse tree of a sentence is shown in Figure 2. The query defines variables a, b, c, and d within the block /ROOT:{ ... } under the if clause where the *paths* are defined w.r.t. the root of the dependency tree. Variables a, b, and c bind to nodes in the dependency tree as given by the paths. For example, a binds to a node that may occur arbitrarily deep under the root node of the tree. The syntax "//" denotes arbitrarily deep. On the other hand, b is defined to be the "*dobj*" node directly under the a node, and c binds to the node with the token "*delicious*" that may occur arbitrarily deep under the b node. For variables d and e, which are defined in the first line of the query under the extract clause, each variable binds to a span of tokens. Outside the block, (b) in (e) is a constraint which asserts that the "*dobj*" (a parse label denoting direct object) token must be among the tokens that make up entity e. The query considers *every* combination of a, b, c, d and e bindings that are possible.

For the sentence in Figure 2, there is only one possible set of bindings for the variables in this query: a = "*ate*", b = "*cream*", c = "*delicious*", d = "*a chocolate ice cream, which was delicious*", and e = "*chocolate ice cream*". The query returns the pair (e,d).

Similarity and aggregation conditions The KOKO language allows for extractions that accommodate variations in linguistic expression and aggregation of evidence over the input. Specifically, it allows for expressions that "softly" match the text and then it aggregates multiple pieces of evidence before yielding an extraction.

Consider the task of extracting cafe names from blog posts. Cafe names are varied and the way cafes are described in different blog posts are also varied. Thus, it would be difficult to write rules that would extract them

accurately from multiple blogs. KOKO combines evidence from multiple mentions in the text and only extracts cafe names that have sufficiently high confidence. For example, if we see that an entity *employs baristas* and *serves espresso*, we may conclude that the combined evidence is sufficient to indicate that the entity is a cafe. However, if it only *employs baristas*, we may decide against that conclusion. In scouring for evidence, care is needed to accommodate linguistic variations on how these properties are expressed, such as *serves up delicious cappuccinos*, or *hired the star barista*. KOKO includes a semantic similarity operator that retrieves phrases that are linguistically similar to the one specified in the rule. Semantic similarity can be determined using paraphrase-based word embeddings. KOKO attaches a confidence value to the phrases matched by the similarity operator, and these confidence values can be aggregated from multiple pieces of evidence in a document. For example, if we see that an entity *serves great macchiatos* and *recently hired a barista*, that may match well to our conditions *serves espressos* and *employ baristas* respectively.

Figure 4 shows an example KOKO query where the aggregation conditions are specified under the satisfying clause (the if clause of the query is empty in this example). The first 3 conditions each have weight 1 while the last two conditions have weight 0.5 each. Intuitively, the weights specify how important a condition is to the overall collection of evidence. Aggregation conditions are either boolean conditions or *descriptors*. For example, the first three conditions are boolean conditions where the first two check whether the string x contains the specified string ("*Cafe*" or "*Roasters*") or not and the last one checks whether x is followed by the string "*, a cafe*" in the text. The remaining conditions are *descriptors* that check whether the span x is followed by the phrase similar to "*serves coffee*" or "*employ baristas*". Descriptors accommodate the fact that different blogs are likely to describe cafes differently. Hence, it is highly unlikely that we will find exact matches of the phrase "*serves coffee*", which is why the descriptor conditions play an important role. The excluding condition ensures that x does not match the string "*La*" (or "*la*") "*Marzocco*", which refers to an espresso machine manufacturer.

To summarize, a basic KOKO query has the following form, where there can be up to one satisfying clause for each output variable.

extract *output tuple* from *input.txt* if
  variable declarations, conditions, and constraints
[satisfying *output variable*
  conditions for aggregating evidence
with threshold α]
[excluding *conditions*]

For every sentence in input.txt, if the extract clause is satisfied, KOKO will search input.txt to compute a score for every satisfying clause. It does so by computing, for every sentence, a score that reflects the degree of match according to the conditions and weights in the satisfying clause and then aggregating the scores of the sentences. For every variable, if the aggregated score of the satisfying clause for that variable from the collective evidence passes the threshold stated, then the result is returned.

More precisely, the score of a value e for the variable under consideration is the weighted sum of confidences, computed as follows: score(e) = $w_1 * m_1(e) + \ldots + w_n * m_n(e)$ where w_i denotes the weight of the i-th condition and $m_i(e)$ denotes the degree of confidence for e based on condition i. The confidence for e is computed for each sentence in the text and aggregated together.

While the idea of specifying extractions by leveraging dependency trees has been explored in the past [35, 36], KOKO is novel in that it provides a single declarative language that combines surface-level and tree patterns, aggregates and combines evidence, and uses novel indexing techniques to scale to large corpora.

Multi-indices for the text KOKO maintains several indexes on the text to process queries efficiently. The indices are built offline, created when the input text is first read. Indices can be persisted for subsequent use.

There are two types of indices in KOKO: *inverted index* and *hierarchy index*. We create *inverted indices* for words and entities and *hierarchy indices* for parse labels and POS (Part-Of-Speech) tags. Unlike indices of [5, 17] our hierarchy index is a compressed representation over dependency structure for parse labels and POS tags. By merging identical nodes, our hierarchy index reduces more than 99.7% of the nodes for both parse labels and POS tags. Hierarchy indices are therefore highly space efficient and enable fast searching.

Intuitively a *word/entity index* maps words/entities to sentences that contain them along with relevant meta-

data. For a word index, every word points to the sentences that contain them, along with the token id and the first and last token id of the subtree rooted at the current token based on the dependency tree, and the depth of the token in the dependency tree. An entity index is defined similarly but the metadata contains information on the span of tokens that constitute the entity. The entity indices enable fast access to sentences that contain certain words and one can also reason about the ancestor-descendant relationship through the metadata that is stored.

A *hierarchy index* is a compact representation of all dependency trees, which provides fast access to the dependency structure of all sentences. There are two types of hierarchy indices: PL (parse label) hierarchy index and POS index. A hierarchy index is constructed by merging identical nodes of dependency trees of all sentences together. Starting at the root of all dependency trees, children nodes with the same label are merged, and then for each child node, all children nodes with the same labels are merged and so on. Hence, by construction, every node in a hierarchy index has a set of children nodes with distinct labels. Consequently, every node of the index can be identified through a unique path given by the sequence of labels from the root node to that node. Every node is annotated with a posting list, which tracks tokens of sentences that have the given path.

We leave the details of how these indices are exploited to [38]. Our experiments demonstrate that the indices are compact and enable us to speed up query processing by at least a factor of 7.

It is important to contrast KOKO with machine learning approaches for recognizing occurrences of typed entities in text (e.g., [8, 28]). Machine learning models are typically specific to a particular domain and do not easily generalize to other domains (e.g., to extract restaurant names instead of cafe names). Also, learning methods usually require significant training data to obtain a model with sufficient precision and recall. In contrast, KOKO does not require training data but relies on user-defined conditions and KOKO is *debuggable* where users can discover the reasons that led to an extraction, which is important for certain applications [6]. Finally, KOKO can be used to create training data for weak-supervision based machine learning approaches.

3 FlexMatcher

Schema Matching refers to the problem of finding correspondences between the elements of different database schemas. Schema matching is one of the critical steps in the data integration pipeline, and thus has received a lot of attention by researchers in the database community (e.g., see [4, 33, 34]). Many of the proposed solutions for schema matching use machine-learning techniques to find the correct correspondences between the input schemas. FLEXMATCHER is an open-source implementation of such a solution in Python. More specifically, FLEXMATCHER is a system for matching the schemas of multiple data sources to a single mediated schema. FLEXMATCHER uses a collection of supervised machine-learning techniques to train a model that can predict the correct correspondence between the schema of a new data source and the mediated schema. The idea is that after observing a small set of data sources that have been manually mapped to the mediated schema, FLEXMATCHER should be able to predict the correspondences for a new data source. For the most part, FLEXMATCHER is an adaptation of the LSD system proposed by [9]. We provide a brief description of FLEXMATCHER's design and operation next.

The architecture of FlexMatcher Figure 5 illustrates the architecture of FLEXMATCHER and describes the training process through a simple working example. Assume that our mediated schema consists of only two attributes: *company name* (associated with color blue) and *salary* (associated with color yellow). Now given multiple datasets listing the salaries of individuals from different companies, our goal is to automatically match the schemas of new datasets to the mediated schema.

As Figure 5 shows, the input to the system is a collection of tables. Note that the input tables can vary in size both in terms of the number of columns as well as the number of available data points. Each table used for training FLEXMATCHER should be accompanied by its correspondences from its columns to the mediated schema. These correspondences are demonstrated with colored arrows on the input tables in Figure 5. For instance, the first two columns in the second table are mapped to attributes company, salary while the last

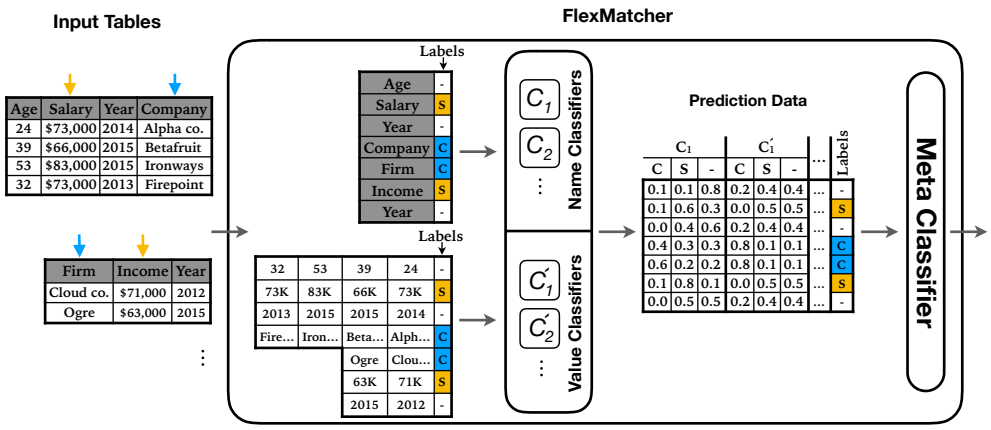


Figure 5: FLEXMATCHER's architecture and training process of the base classifiers and meta-classifier.

column does not map to any attribute of interest in the mediated schema.

Given the input tables, FLEXMATCHER then creates two training datasets. The first dataset (placed on top) consists of the names of columns along with their associated correspondences. In contrast, the other dataset only contains the values under each column along with their associated correspondences. These two training datasets are then fed to an arsenal of classifiers which we refer to as *base* classifiers. Base classifiers are responsible for both extracting relevant features as well as building a model for predicting the correspondences. Note that the base classifiers are divided into two groups: classifiers on column names and classifiers on column values. Naturally, the *name classifiers* are efficient at extracting features from the text and detecting text similarities, while the *value classifiers* are more suitable for extracting features from a collection of values that can be *string*, *numeric*, or *categorical*. Value classifiers can even extract features based on the distribution of the input values (such as min, max, mean, and etc.) to make accurate predictions.

Given that FLEXMATCHER uses a collection of base classifiers, it needs to determine how to combine the prediction of all classifiers to reach a final prediction. To achieve this, FLEXMATCHER first uses each base classifiers to predict the training data using a 5-fold cross-validation technique. This means the data is sliced into 5 folds, and then each fold is predicted using the other folds as training data for the classifier. The output of this step, is the probability distribution over the attributes in our mediated schema for each column. The probability distributions reported by each base classifiers are then organized into a table which is marked as *prediction data* in Figure 5. Finally, FLEXMATCHER trains a *meta-classifier* on the prediction data. The meta-classifier is a simple *logistic regression* model that learns how to combine the prediction of each base-classifier.

Once the training process is complete, FLEXMATCHER can make predictions for new data sources by (1) making prediction based on column names using name classifiers, (2) making predictions based on data points via value classifiers, and (3) combining the prediction of all base-classifiers using the pre-trained meta classifier.

Next, we highlight some aspects of the design of FLEXMATCHER that allows practitioners to easily modify the tool to improve the performance in different settings.

One of the most important feature of FLEXMATCHER is that it allows the user to add/remove base classifiers (of either type) as they see fit. This enables practitioners to both benefit from the existing classifiers in FLEXMATCHER while having the option to add classifiers that can work with features that are expected to be more helpful in a particular domain. For instance, if we know that in our setting the salaries are always reported with the currency symbol (such as $ or €), then we can simply add a new classifier that predicts a column as "salary" if it can detect any currency symbol in the data. To make the process of adding and removing classifiers easy, FLEXMATCHER is designed to be compatible with python's machine learning ecosystem. More precisely, all the base-classifiers are expected to comply with python's *scikit-learn* package [30] which implements a great

number of machine learning techniques. Thus, one can easily deploy different models from the scikit-learn package and use them as part of FLEXMATCHER.

Following our last example, it is important to mention that base classifiers do not need to be efficient at detecting all attributes in the mediated schema. Clearly, the "salary" classifier we just proposed can not distinguish between "company name" or "last name". Nevertheless, the meta classifier learns to trust this classifier if it predicts a column as "salary" and ignore its predictions in other scenarios. As a result, users can easily build new classifiers that are focused and tailored to a single attribute, and thus enhance the overall performance of FLEXMATCHER in their setting.

FLEXMATCHER also allows users to enforce cardinality constraints on the predicted correspondences. For instance, we can enforce that at most a single column should be mapped to each of the attributes in the mediated schema. These constraints are handled by solving a *maximum bipartite-matching* problem. More precisely, the goal is to match the columns to the attributes in the mediated schema such that (a) the cardinality constraints are met, and (b) the sum of FLEXMATCHER's confidence over all mapped attributes is maximized. To achieve this, FLEXMATCHER applies the Hungarian algorithm to find the optimal matching.

Finally, FLEXMATCHER has the ability to cope with absence of values in the input data-sources. In this case, the system only deploys the name classifiers to make a prediction based on the common patterns that it observes in column names. Alternatively if the data source miss column names, then FLEXMATCHER would only deploy the value classifiers to predict the correspondences. This makes FLEXMATCHER more robust to such data-quality issues.

4 Entity Matching Packages

4.1 String Similarity Measures and Similarity Joins

String matching is a fundamental operation in many data science tasks, such as data exploration, data profiling, data cleaning, information extraction, schema and entity matching. As a result, over the past few decades, string matching has received significant attention and many solutions have been proposed [11, 39]. At present, BIGGORILLA contains two packages related to string matching: PY_STRINGMATCHING and PY_STRINGSIMJOIN.

PY_STRINGMATCHING: Given two sets of strings A and B, *string matching* is the problem of finding all pairs $(a \in A, b \in B)$ that match, i.e., refer to the same real-world entity, such as "Michael J. Williams" and "Williams, Michael". So far the most common solution is to return as matches all pairs $(a, b) \in A \times B$ that satisfy a predicate of the form $sim[t(a), t(b)] \geq \epsilon$. Here $sim(a, b)$ is a string similarity measure, t is a tokenizer, and $\epsilon \in [0, 1]$ is a pre-specified threshold. For example, given the predicate *jaccard*[*3gram*(a), *3gram*(b)] > 0.8, we tokenize strings a and b into sets of 3-grams S_a and S_b respectively, compute the Jaccard score $|S_a \cap S_b|/|S_a \cup S_b|$, then return true if this score exceeds 0.8 and return false otherwise.

Numerous string similarity measures have been developed, such as Jaccard, edit distance, TF/IDF, cosine, etc. [11, 39]. PY_STRINGMATCHING implements a broad range of these similarity measures, together with a set of well-known tokenizers. The rationale for developing this package and a comparison with nine existing string similarity measure packages are provided in the package's developer manual [16].

PY_STRINGSIMJOIN: This package performs a *string similarity join* between two sets A and B, using a predicate such as *jaccard*[*3gram*(a), *3gram*(b)] > 0.8 as the join condition [39]. Applying the join predicate to all pairs in $A \times B$ is often impractical because $A \times B$ can be very large. To address this, current work typically builds an index I over a table, say A (sometimes multiple indexes are built, over both tables). For each string $b \in B$, it then consults I to locate only a relatively small set of strings in A that can potentially match with b. For example, suppose I is an inverted index that, given a token t, returns the IDs of all strings in A that contain t. Then for a string $b \in B$, we can consult I to find only those strings a in A that share at least a token with b, then apply the join predicate only to these (a, b) pairs. Numerous such indexing techniques have been developed,

e.g., inverted index, size filtering, prefix filtering, etc. [11, 39]. The package PY_STRINGSIMJOIN implements a variety of string similarity joins using these indexing techniques. Together, these two packages provide a basis for us to build more advanced packages, such as MAGELLAN, a package to perform end-to-end entity matching, which we describe next.

4.2 End-to-End Entity Matching with Magellan

The MAGELLAN system focuses on entity matching (EM), the problem of identifying data instances that refer to the same real-world entity, such as (David Smith, UW-Madison) and (D. M. Smith, UWM). This problem has been a long-standing challenge in data management [7, 13]. MAGELLAN introduces a new template for research and system development for EM.

The current research template for EM focuses largely on developing algorithmic solutions for *blocking* and *matching*, two important steps in the EM process. However, a recent paper [17] argues that EM processes often involve many other "pain points", and proposes a new research template in which we move beyond examining just blocking and matching. Instead, we would (a) develop a step-by-step how-to guide that captures the entire EM process, (b) examine the guide to identify all true pain points of the EM process, then (c) develop solutions and tools for the pain points. Several recent works [9, 20] confirm that there are indeed many other pain points including data labeling, debugging, EM-centric cleaning, and defining the notion of match. Furthermore, the research argues that solving these pain points is critical for developing practical EM tools, and that addressing them raises many novel research challenges.

Most current EM systems are built as *stand-alone monolithic systems*. However, in [17] we argue that such systems are very difficult to extend, customize, and combine. We observe that many EM steps essentially perform data science tasks, and that there already exist vibrant ecosystems of open-source data science tools (e.g., those in Python and R), which are being used heavily by data scientists to solve these tasks. Thus, we propose to *develop EM tools within such data science ecosystems*. This way, the EM tools can easily exploit other tools in the ecosystems, and at the same time make such ecosystems better at solving EM problems.

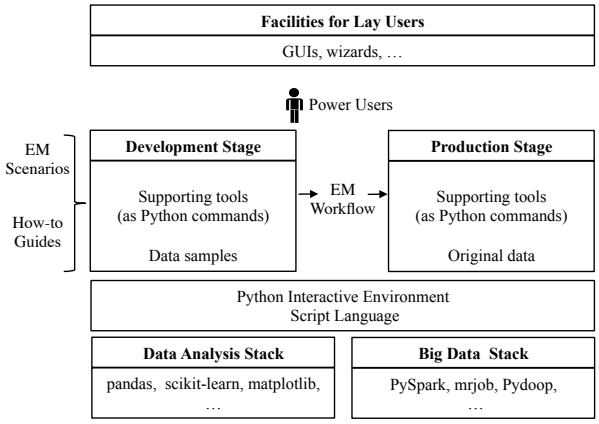


Figure 6: The MAGELLAN architecture.

Figure 6 shows the MAGELLAN architecture. The system targets a set of EM scenarios. For each EM scenario it provides a how-to guide. The guide proposes that the user solve the scenario in two stages: development and production. In the development stage, the user develops a good EM workflow (e.g., one with high matching accuracy). The guide tells the user what to do, step by step. For each step which is a "pain point", the user can use a set of supporting tools (each of which is a set of Python commands). This stage is typically done using data samples. In the production stage, the guide tells the user how to implement and execute the EM workflow on the entirety of data, again using a set of tools.

Both stages have access to the Python interactive scripting environment (e.g., Jupyter Notebook). Since tools are built into the Python ecosystem, MAGELLAN can borrow functionalities (e.g., cleaning, extraction, visualization) from other Python packages in the ecosystem. Finally, the MAGELLAN is currently geared toward users who are knowledgeable in programming. In future facilities for lay users (e.g., GUIs, wizards) can be provided so that lay user actions can be translated into sequences of commands for MAGELLAN.

We describe the capabilities of MAGELLAN next, which shows that it goes beyond blocking and matching.

- *Loading tables:* MAGELLAN relies on the pandas package to read tables on file into data frames. As a result, it can read a wide variety of files, including csv, json, XML, etc.

- *Browsing, exploration, profiling, and cleaning:* MAGELLAN uses the pandas-profiling, matplotlib, and other popular Python packages to help the user visualize and profile the tables. It can connect to OPENREFINE, a stand-alone data exploration and cleaning tool, to help the user explore and clean the tables.

- *Downsampling and blocking:* If two tables A and B to be matched have been read into memory and they are big, then the user can downsample them into smaller tables A' and B', which would be easier to work with. MAGELLAN also provides the user with a variety of blocking methods (e.g., attribute equivalence, overlap, rule-based, etc.). More blocking methods are being developed. Finally, MAGELLAN also provides a blocking debugger, which can be used to examine how good the current blocker is [24].
- *Feature generation:* Both the blocking step and the matching step (described later) use a set of features, such as *edit_dist*(A'.*city*, B'.*city*) and *jaccard*[*3gram*(A'.*name*), *3gram*(B'.*name*)]. MAGELLAN uses the two packages PY_STRINGMATCHING and PY_STRINGSIMJOIN described in the previous section to create these features. We are also currently working on helping lay users learn blocking rules.
- *Sampling and labeling:* After blocking, the user can take a sample of tuple pairs and label them as match/no-match. MAGELLAN provides several labeling tools, including a Web-based tool that enables collaborative labeling. We are currently working on a tool to help debug the labels.
- *Selecting and debugging the matchers:* Using the labeled data, the user selects a learning-based matcher (from the library of learning methods in scikit-learn), evaluate its accuracy, and debug if necessary. MAGELLAN currently provides tools to debug decision trees and random forests. We have also recently developed a deep learning based matcher that works well for textual and dirty tuples [27] (see the next section).
- *Clustering and merging matches:* Once a matcher has been selected, the user can apply it to the tuple pairs in the output of the blocker to predict matches. We are currently working on tools to cluster the tuples (such that all tuples within a single cluster match) and to merge tuples within each cluster into a single canonical tuple.
- *Adding matching rules to the workflow:* MAGELLAN also allows the user to easily add matching rules to various steps in the EM workflow.

As described, MAGELLAN relies heavily on packages in the Python ecosystem. In addition to developing the above capabilities, we are also developing methods and tools to scale up the execution of MAGELLAN commands, to manage data larger than memory, to specify EM workflows, to perform incremental execution of workflows, and to track provenance, among others. Finally, MAGELLAN provides detailed guidance to users on the challenges that commonly arise during EM, such as how to collaboratively label a dataset (so that the labelers can quickly converge to a match definition), how to decide when to stop the blocking step, and how to determine the size of the sample to be labeled.

4.3 Deep Learning for Semantic Matching Tasks

The entity matching problem discussed in the previous section is just one case of a *semantic matching* problem. In the above case, matching means that the two data instances refer to the same real world entity. A different example of semantic matching is answer selection (a major sub-task of question answering), which determines if a candidate sentence or paragraph correctly answers a given question. Here the comparison is between two natural language text blobs, a question and an answer candidate. Other types of semantic matching tasks include textual entailment, entity linking, and coreference resolution.

Deep learning (DL) solutions have been shown to achieve state-of-the-art performance for these tasks [37, 15]. While DL models solutions for these tasks seem highly specialized at first glance, they do in fact share several commonalities. Based on a comprehensive review [27], we note that all of these models share the following major components:

- *Language representation:* This component maps words in each data instance into word embeddings, i.e., vector representations of words. For example, it would convert a question into a sequence of vectors.
- *Similarity representation:* This component automatically learns a vector representation that captures the similarity of the two data instances given their language representations. This involves summarizing each data instance and then comparing them.
- *Classification:* This component takes the similarity representations and uses those as features for a classifier that determines if the two data instances match.

These similarities present a great opportunity - a system that can address one of these tasks can easily tackle other matching tasks. However, developing such a system is a cumbersome endeavor. Even when using popular deep learning frameworks, many aspects such as data processing, training mechanics, model construction, etc. need to be handled manually. To simplify building deep learning models for semantic matching tasks, we developed DEEPMATCHER, a Python package based on the model architecture described above. It is broadly applicable to matching tasks such as entity matching, question answering, textual entailment, DEEPMATCHER is built on top of PyTorch [29], a deep learning framework that is great for research as it enables rapid development iterations. Once a model has been finalized and trained, it can then be optimized for fast inference in production settings.

In order to help users get started, DEEPMATCHER comes built-in with four representative DL models, based on four major categories of neural networks used in matching tasks [27]. The four models vary in complexity and provide a trade-off between training time and model expressiveness:

- *SIF:* This model first aggregates information about the words present in each data instance, and then compares them. It does not take word sequence into account. Intuitively, data instances that contain similar words are considered matches.
- *RNN:* This model first performs a sequence-aware summarization of each data instance, and then compares them. Intuitively, data instances that contain similar word sequences are considered matches.
- *Attention:* This model first performs an alignment between words in each data instance, then performs a word-by-word comparison based on alignment, and finally aggregates this information to perform classification. Intuitively, data instances that contain *aligning words* are considered matches.
- *Hybrid:* This model first performs an alignment between word sequences in each data instance, then compares the aligning word sequences, and finally aggregates this information to perform classification. Intuitively, data instances that contain *aligning word sequences* are considered matches.

While DEEPMATCHER provides these four representative solutions out of the box, the package is designed to be easily extensible. It provides several options for each component described earlier. Users can create their own models using the provided building blocks and optionally use custom modules. The four built-in models can also be thoroughly customized. It is built to be both simple enough for deep learning beginners and also highly flexible for advanced users.

There are four main steps in using DEEPMATCHER. First, the user preprocesses all data instances. This involves tokenization, loading word embeddings, etc. Second, the user constructs a neural network model for matching. Third, the user trains the neural network model on the preprocessed training dataset. Finally, the user may apply the trained model over the test set or use it to make predictions. Without customization, this entire pipeline requires less than 10 lines of code. A version of this package was used to perform an extensive empirical evaluation of deep learning approaches for the task of entity matching, compared to MAGELLAN [27]. Apart from entity matching, DEEPMATCHER can be used for other semantic matching tasks, such as answer selection for question answering, and textual entailment.

5 Applications

BigGorilla has been used in several projects within Recruit Holdings. Some examples include mining of restaurant/salon names, matching schemas of real estate records, and entity extraction from textual statements. In addition, there are tutorials on various components of BigGorilla that can be used as course assignments for a data science course.

In one of the applications, KOKO was used to extract names of companies from text and the result was fed into Magellan to determine entity matching across two datasets. Prior to KOKO, extraction of company names was carried out manually and with the help of KOKO, the amount of human labor is reduced and some work is expended instead to verify whether the entity names are extracted correctly.

FlexMatcher has been used to match schemas of real estate records from different agencies. FlexMatcher semi-automates the schema matching process with data-driven approaches, and improves accuracy by identifying a number of correct matching instances missed by human workers. For example, FlexMatcher successfully matched two columns containing similar data – i.e., "*mls_subpremise*" and "*L_Address2 Unit #*" – although the syntax of the column names are quite different.

Magellan has been used in several projects at Recruit Holdings to help improve the accuracy and efficiency of their entity matching tasks [1]. For example, one restaurant-advertising company collects restaurant information (e.g., reviews and ratings) from online resources regularly. The collected information is then consolidated with an existing restaurant database. Magellan is used to match between restaurant names, which reduces the cost of matching significantly, with 4% increase in entity matching accuracy at the same time. In other projects, Magellan has also been used to match salon/nail shop names, with similar performance improvement.

BigGorilla also contains tutorials which showcase key components of the data integration pipeline and can be used for educational purposes. Currently, there are tutorials that showcase roughly 3 steps (extraction, schema matching, and entity matching and some cleaning) of the data integration pipeline. These tutorials illustrate KOKO, FlexMatcher and Magellan through an example problem that analyzes aviation data from multiple data sources, to produce statistics, such as the airlines that have the most accidents in history. These tutorials are freely available for use and can be easily adapted to be used as course assignments for educational purposes.

The packages PY_STRINGMATCHING, PY_STRINGSIMJOIN, and MAGELLAN have also been successfully used in five domain science projects at UW-Madison (in economics, biomedicine, environmental science [1, 18, 21, 30]), and at several other companies (e.g., Johnson Control, Marshfield Clinic, WalmartLabs). For example, at WalmartLabs they improved the recall of a deployed EM solution by 34%, while reducing precision slightly by 0.65%. They have also been used by 400+ students to match real-world data in five data science classes at UW-Madison (e.g., [2]).

6 Conclusion

The goal of BIGGORILLA is to become a one-stop open source resource for data scientists working on data preparation and integration, and grow an active community supporting this effort. We have described four software packages (KOKO, FLEXMATCHER, MAGELLAN, and DEEPMATCHER) which we have contributed to BIGGORILLA. We hope that as more software packages are added to BIGGORILLA, it will further benefit both researchers and industry practitioners in understanding the state-of-the-art in data preparation and integration, what is freely available for use, and perhaps more importantly, what more can be done.

References

[1] BigGorilla: An Open-source Data Integration and Data Preparation Ecosystem: https://recruit-holdings.com/news_data/release/2017/0630_7890.html.

[2] CS 838: Data Science: Principles, Algorithms, and Applications https://sites.google.com/site/anhaidgroup/courses/cs-838-spring-2017/project-description/stage-3.

[3] M. Bernstein et al. MetaSRA: normalized human sample-specific metadata for the sequence read archive. *Bioinformatics*, 33(18):2914–2923, 2017.

[4] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011.

[5] S. Bird, Y. Chen, S. B. Davidson, H. Lee, and Y. Zheng. Designing and evaluating an xpath dialect for linguistic queries. In *ICDE*, page 52, 2006.

[6] L. Chiticariu, Y. Li, and F. R. Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *EMNLP*, pages 827–832, 2013.

[7] P. Christen. *Data Matching*. Springer, 2012.

[8] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, 2011.

[9] A. Doan, P. Domingos, and A. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Mach. Learn.*, 50(3):279–301, Mar. 2003.

[10] A. Doan et al. Human-in-the-loop challenges for entity matching: A midterm report. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2017, Chicago, IL, USA, May 14, 2017*, 2017.

[11] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Morgan Kaufmann, 1st edition, 2012.

[12] X. L. Dong and D. Srivastava. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.

[13] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, 2007.

[14] FlexMatcher (Schema Matching package in Python). https://github.com/biggorilla-gh/flexmatcher,

[15] M. Francis-Landau, G. Durrett, and D. Klein. Capturing semantic similarity for entity linking with convolutional neural networks. NAACL, 2016.

[16] P. S. G.C. et al. Py_stringmatching's developer manual, pages.cs.wisc.edu/˜anhai/py_stringmatching/v0.2.0/dev-manual-v0.2.0.pdf, 2017.

[17] S. Ghodke and S. Bird. Fast query for large treebanks. In *HLT-NAACL*, pages 267–275, 2010.

[18] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W. Tan. Data integration: After the teenage years. In *PODS*, 2017.

[19] Google Cloud Natural Language API. https://cloud.google.com/natural-language/,

[20] Y. Govind et al. Cloudmatcher: A cloud/crowd service for entity matching. In *KDD Workshop on Big Data as a Service (BIGDAS-17)*, 2017.

[21] P. Konda et al. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[22] P. Konda et al. Performing entity matching end to end: A case study. 2016. Technical Report, http://www.cs.wisc.edu/ anhai/papers/umetrics-tr.pdf.

[23] E. LaRose et al. Entity matching using Magellan: Mapping drug reference tables. In *AIMA Joint Summit*, 2017.

[24] H. Li et al. Matchcatcher: A debugger for blocking in entity matching. In *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018.*, 2018.

[25] R. T. McDonald, J. Nivre, Y. Quirmbach-Brundage, Y. Goldberg, D. Das, K. Ganchev, K. B. Hall, S. Petrov, H. Zhang, O. Täckström, C. Bedini, N. B. Castelló, and J. Lee. Universal dependency annotation for multilingual parsing. In *ACL*, pages 92–97, 2013.

[26] R. J. Miller. The future of data integration. In *KDD*, pages 3–3.

[27] S. Mudgal et al. Deep learning for entity matching: A design space exploration. In *SIGMOD-18*, 2018.

[28] D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Lingvisticæ Investigationes*, 30(1):2–36, 2007.

[29] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] P. Pessig. Entity matching using Magellan - Matching drug reference tables. In CPCP Retreat 2017. http://cpcp.wisc.edu/resources/cpcp-2017-retreat-entity-matching.

[32] S. Petrov, D. Das, and R. T. McDonald. A universal part-of-speech tagset. In *LREC*, pages 2089–2096, 2012.

[33] E. Peukert, J. Eberius, and E. Rahm. A self-configuring schema matching system. In *ICDE*, pages 306–317, 2012.

[34] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[35] Tregex, tsurgeon, semgrex. http://nlp.stanford.edu/software/tregex.html,

[36] M. A. Valenzuela-Escárcega, G. Hahn-Powell, and M. Surdeanu. Odin's runes: A rule language for information extraction. In *LREC*, 2016.

[37] S. Wang and J. Jiang. A compare-aggregate model for matching text sequences. ICLR, 2017.

[38] X. Wang, A. Feng, B. Golshan, A. Halevy, G. Mihaila, H. Oiwa, and W.-C. Tan. Scalable semantic querying of text *(to appear)*. *PVLDB*, 2018.

[39] M. Yu et al. String similarity search and join: A survey. *Front. Comput. Sci.*, 10(3):399–417, 2016.

Self-Service Data Preparation: Research to Practice

Joseph M. Hellerstein, Jeffrey Heer and Sean Kandel
{joe, jheer, skandel}@trifacta.com

1 The Arrival of the Category

It is widely accepted that the majority of time in any data analysis project is devoted to preparing the data [25]. In 2012, noted data science leader DJ Patil put the fraction of time spent on data preparation at 80%, based on informal discussions in his team at LinkedIn [28]. Analysts we interviewed in an academic study around the same time put the percent time "munging" or "wrangling" data at "greater than half" [19]. Judging from these user stories, the inefficiency of data preparation is the single biggest problem in data analytics.

The database community does have a tradition of research on related topics. Most common is algorithmic work (covered in various surveys, e.g. [15, 4, 3, 26]) that focuses on automating certain aspects of data integration and cleaning, or that uses humans as passive computational elements via crowdsourcing.

What computer science researchers often miss are the skills realities in real-world organizations. In practice, the primary bottleneck is not the quality of the inner-loop algorithms, but rather the lack of technology enabling domain experts to perform end-to-end data preparation without programming experience. Fully preparing a dataset requires an iterated cycle of input data quality assessment, transform specification, and output quality assessment—all in service of a larger "preparedness" goal that tends to shift fluidly as the work reveals additional properties of the data. Traditionally there has been a divide between the people who know the data and use case best, and the people who have the skills to prepare data using traditional programmatic approaches. This results in the data preparation cycle being split across parties and across time: domain experts try to express their desired outcomes for prepared data to developers or IT professionals, who in turn try to satisfy the needs. A single iteration of this cycle can take from hours to weeks in a typical organization, and rarely produces a satisfying outcome: typically either the end-user did not specify their desires well, or the developer did not achieve the desired outcome. Neither tends to enjoy the experience. In short, the primary problem in data preparation is *self-service*: we need to enable the people who know the data best to prepare it themselves.

Research focused on these user-facing concerns is scattered across the fields of databases, HCI and programming languages (e.g., [7, 29, 13, 24, 17, 9, 16]). While under-investigated in the research community, the topic has become an important force in the industry. In 2015, industry analysts began publishing rankings in an emerging new market category dubbed "Self-Service" or "End-User" Data Preparation [5, 1]. Two years later, the established analyst firm Forrester did a first annual Forrester Wave report on Data Preparation [23]: a mark of arrival for this category. Another major analyst firm, Gartner, has weighed in with various reports on the Data Preparation market (e.g. [35, 8]). Meanwhile, in 2017 Google Cloud Platform was the first cloud provider to launch Self-Service Data Preparation as a native service in their cloud [27], while Azure and Amazon Web Services also announced partnerships with data preparation vendors in 2018. Market size estimates for Data Preparation start at $1 billion [35] and go well upwards depending on the analyst and projected time horizon. Not bad for a technology that was seeded from academic research, and did not even have a name four years ago.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In this article we share some of our experience navigating the path from research to commercial products in this domain, which we and our users at Trifacta affectionately refer to as Data Wrangling. We begin by providing a snapshot of the space as we are seeing it, including motivating customer use cases, and distinctions between the new Data Wrangling market and co-existing older markets in Data Integration and ETL. Then we go back to the research roots of Trifacta and talk about how our views of users and technical requirements have evolved.

1.1 A Data Wrangling Case Study

At Trifacta, we have seen Data Wrangling use cases in a wide variety of contexts and markets; any organization that does data analysis needs to do data preparation. There is broad usage in traditional industries like financial services, telecommunications, pharmaceuticals and health care. But we also see interesting use cases in everything from assembling national voter rolls for the 2016 presidential election to decoding data from drones.

As a case study, consider the following example usage in public health. In late 2014 there was surge of HIV cases in rural Scott County, Indiana, which eventually infected nearly 200 people. The US Center for Disease Control (CDC) came in to study patients and assemble a large number of data sources, and found that sharing of opioid needles was the primary vector for infection. Their recommendation for the best way to stop the spread of AIDS in Scott County: establish clean needle exchanges for drug users. The governor of Indiana at the time, Mike Pence, was "morally opposed to needle exchanges on the grounds that they supported drug abuse". However, after the CDC made its recommendation, the governor said "I'm going to go home and pray on it". In the end, Gov. Pence reportedly "found the science convincing", and approved the needle exchanges. This was 21 months before Pence became Vice President of the US. [33].

During the investigation, the CDC worked under time pressure with a number of disparate data sets including "HIV outbreak clusters, geographic factors, epidemiological patterns, and drug resistance data, among others" [34]. They brought in a systems integrator called Leidos who provided a software platform called CAADS that combines visual data technologies from multiple vendors: Trifacta for data preparation, Tableau for charting, and Alpine Data and Centrifuge Analytics for analytics. Leidos staff reported that Trifacta substantially reduced time for wrangling. "Work that would have taken six weeks now can take as little as a day" [14]. CDC researchers also noted that Trifacta enabled them to detect missing records and outliers that would have otherwise polluted the analysis, and that were overlooked by prior data cleaning tools [22].

The CDC is now recommending that state health departments prepare and monitor data from an increasingly large pool of sources on an ongoing basis, "to identify jurisdictions that, like this county in Indiana, may be at risk of an IDU-related HIV outbreak. These data include drug arrest records, overdose deaths, opioid sales and prescriptions, availability of insurance, emergency medical services, and social and demographic data" [2].

Some of the key–and archetypal–features of this example include the following:

Self Service: The data was being acquired, prepared and analyzed not by computer experts, but by the domain experts who understood the data best: public health experts in the field. "CAADS is very much built to ... support the range of public health aspects and to allow people who would otherwise not be able to get into data analytics to have access and do so in a responsible way." [34]

Agility: The users were engaged in rapid collaboration with visual tools to explore multiple hypotheses in a lightweight iterative fashion. "They would be on a call discussing the outbreak and someone would say 'What if we did it this way?' They'd step away for a minute, run the analysis, and say, 'No that didn't work. But if we do it this other way, here's the result"' [34].

Veracity and Explainability: The results of the study were to be presented for executive decision-making with an expectation of critical feedback. In that setting, intuitive explanations of recommendations were critical.

More Data, More Data Sources: The Indiana effort involved a wide range of data, acquired quickly from a variety of source in a variety of ways. Looking ahead, the CDC expects this set of sources to grow, and sees a need to monitor feeds from these sources over time.

2 Key Tasks in Data Wrangling

Here we briefly overview the main tasks in wrangling data. For a more thorough treatment we refer the reader to a book on the subject [32].

Unboxing: Discovery & Assessment: If you have worked with new data you know that the first step is the wonder of unboxing: what's in there? Basic questions arise right away. How is the data structured: does it have discrete records and fields? Is it flat or nested, uniform or irregular, dense or sparse? Is there metadata embedded in the data that should be pulled out? What types of fields are there, and how are they coded? Do particular values of interest occur in the data...am I in the data? What are the data distributions: the popular and rare categorical variables, the distribution of numerical variables? You cannot begin thinking about data analysis before understanding the answers to these basic questions. Visualization can help a great deal here, but note that only the last questions above match typical charting packages and BI. The other questions relate more closely to parsing and second-order schematic transformations that involve manipulating both data and metadata.

Structuring. Analytics software likes its data in rectangles: tables or matrices. Different rectangles fuel different analyses: sometimes a matrix or pivot table is the right thing, other times a relational table is better. As a result, it is common to restructure data whether or not it arrives structured. Typical (re)structuring transformations include pivoting tables into matrices and vice versa, as well as unnesting key/value sets ("dictionaries", "maps" or "objects") or arrays/lists into columns. These transforms have the property that they convert data (say keys in a JSON document) into metadata (column names) or vice versa: they are second order data transformations unfamiliar from SQL and logic. Users tend to find these transformations very non-intuitive, so various visual metaphors are often used to help users route values to appropriate coordinates in a grid.

Cleaning. Neat is not the same as clean: tabular or matrix data can often be dirty in the sense of failing to capture expectations. One class of expectations capture the notion of "central" and "outlying" values. Outliers can be statistical (e.g. distance from the center of a distribution) or semantic ('Italy' is the not the name of a US state.) Another class of expectations are logical: boolean properties like functional dependencies. Yet another class relates to encoding: e.g., in entity resolution, objects with multiple categorical labels need to have their labels mapped together. Expectation checking for data cleaning often involves consideration of individual records with respect to a larger dataset or population, which can be difficult for humans to do manually—algorithms are needed. But the outputs of these algorithms are often uncertain or ambiguous, and human intervention can be required to complete the job. The surveys mentioned earlier provide overviews of these issues [15, 4, 3, 26], but some of the hardest problems are in enabling users to assess and correct algorithm outputs.

Enriching & Blending. Data is rarely self-contained; it often benefits from or requires additional data for further context. As a simple example, unresolved "reference" or "foreign key" identifiers need to be decoded by joining to a lookup table. Softer issues also drive the need for data enrichment: for example, to assess the likelihood of the data you have, you may want to compare it to a larger sample from a similar or different population. Or to assess the significance of a correlation between two attributes in your data (say, disease occurrence and age) you may want to join in a third column from another data set to establish if there is conditional independence via a confounding variable (say tobacco use). Algebraically, these activities—often called "data blending" in industry marketing material—encompass a range of transformations including joins, unions, and subqueries with aggregation. It is not always easy to figure out how to do these tasks, or to assess if they were done correctly.

Optimizing & Publishing. Once a dataset is structured and cleaned to an analyst's level of comfort, it is often necessary to do additional preparation to produce appropriate output for the next phase of analysis. For example, publishing data to a relational data warehouse may require normalizing a single nested dataset into a number of tables with key/foreign key relationships. The traditional schema mapping problem that occupied the ETL and Data Integration communities during the 1990's can be viewed as a general example of this kind of work. In modern "big data" environments with machine-generated data, it is often necessary to perform data reduction

before visualizing the results in a traditional charting tool: aggregation or sampling of the cleaned data is still required. In other cases, data has to be recoded to suit the specific API of a downstream product for analysis or charting. These tasks can be as demanding and detail-oriented as any of the other steps in the pipeline.

In sum, in 2018 we should all be aware that data is not "truth": it is evidence to be assessed in the context of specific use cases. If your data is familiar, your use case may change—or vice versa—requiring you to rethink some or all of the steps above.

2.1 Isn't this just ETL? Not really.

At a high level, ETL and data preparation solve similar problems: transforming multiple input datasets to some desired output. However, modern data preparation differs from ETL in nearly every basic question of Who, Why, What and Where, resulting in very different requirements for new software solutions.

ETL technologies are targeted at IT professionals (Who), tasked with establishing static data pipelines for reference data (Why), given structured inputs (What) in on-premises datacenters running legacy software (Where). These ETL systems exist to help IT implement data pipelines for downstream consumers. IT receives data requirements from a line of business, and implements and optimizes these pipelines on behalf of those consumers. IT is also responsible for ensuring security and quality of the data used by analysts. The "Who" does not scale well with the rising tide of data: As organizations become more data-driven and the number of consumers and analytic use cases grows, IT can become an unscalable bottleneck in the analysis lifecycle.

Data preparation tools instead focus on enabling analysts in a specific line of business (Who) to design and maintain data preparation recipes that solve problems relevant to the their job success (Why). They typically bring bespoke data from their own processes, and often blend in external data as well (What). In many cases these organizations are working on net new projects, and are given the latitude to set up on new modern infrastructure, e.g. using public clouds or open source technologies (Where).

The difference in the Who drives the need for self-service technology, reflecting the changing nature of data in large organizations. The line of business likes self-service data preparation because they best understand their own requirements, and self-service cuts out time-consuming communication and delivery cycles with IT. IT increasingly likes self-service, because analysts are not burdening them with tedious and ill-specified data provisioning requests. Self-service data preparation enables organizations to scale the number of analysis tasks they can perform beyond what IT could centrally manage, and make the whole process more efficient and satisfying. Moreover, the change in Why often makes these exercises highly valuable: line-of-business use cases typically use data to make more money or reduce costs. This is creative, competitive data work.

The difference in What is not to be minimized. Legacy ETL is designed to handle well-structured data originating from a variety of operational systems or databases. Many ETL tools are entirely schema-driven: users are not shown data when they specify ETL pipelines. This makes it a poor fit to data that is not clearly schematized in one uniform way, or data that needs cleaning as much as it needs republishing. A growing amount of analysis occurs in environments where the schema of data is not defined or known ahead of time. This means the analyst doing the wrangling determines how the data can be leveraged for analysis as well as the schema (structure) to be imposed to perform that analysis. Legacy ETL systems were not optimized for large-scale data or complex raw sources that require substantial restructuring and derivation.

Finally the Where is changing quickly. To be most effective, ETL and data preparation products should both be integrated into an ecosystem of other tools for storage, processing and consumption. Legacy ETL products were designed when most data was on-premises in relational and operational systems. Because the locus of data gravity is moving to big data and cloud environments, many forward-looking data preparation solutions are focused on working well in those environments.

This is not to say that ETL is dead. Like many legacy technologies, it plays a key role in the standard operating procedure of traditional IT. Rather, the point is that data preparation solutions should provide a new

How to respond to the new Who, Why, What and Where. Legacy ETL use cases are already handled adequately, and there is less of a pressing need for new technologies to solve those old problems.

3 Evolutionary Paths

We launched the original Wrangler prototype on an academic website in Fall 2011, and quickly saw thousands of unique users: our first feedback on the importance of the area. Since that time, we have been through many iterations of refining our understanding of the data wrangling problem and the key issues in improving user productivity. Here we share our learning on a number of fronts, from users to interfaces to backend systems.

3.1 Personas

A critical question for any product, but especially in an emerging market, is this: Who are my users? This question can be particularly difficult to answer in the early stages of design, when there are few if any actual users and instead one has to develop *personas* of who the likely intended users are. Personas are "fictional representations and generalizations of ... your target users who exhibit similar attitudes, goals, and behaviors in relation to your product." [6]. In the case of Trifacta, our understanding of who we are designing for has evolved tremendously as we have engaged different user communities and observed industry-wide changes in organizational data analysis roles.

Unsurprisingly, our initial outlook was anchored in the university environment in which we were pursuing research. We were motivated by our own trials and tribulations with data wrangling, as well as those of our academic peers. This orientation was reflected in our early user studies [18, 10]: largely due to convenience, a majority of our study participants were computer science students with some degree of analysis training.

We recognized the limits of studying this population, and set out to conduct an interview study of analysts in industry [19], involving professionals with varied titles such as "business analyst", "data analyst" and "data scientist". We found that our respondents were well-described by three archetypes ("hackers", "scripters", and "application users") that differed in terms of skill set and typical workflows. These archetypes varied widely in terms of programming proficiency, reliance on information technology (IT) staff, and diversity of tasks, but varied less in terms of statistical proficiency. These initial archetypes, and their corresponding tasks and workflows, served as a major touchstone for our early commercialization efforts.

That is not, however, to say that our archetypes were unbiased. The analysts we interviewed at that time spanned a spectrum of programming proficiency, but all worked in dedicated analysis positions. Increasingly we see people working with data when that is not their primary job role: our case study in Section 1.1 is an example. In addition, we have observed industry-wide shifts in six years as "data science" has matured. While early data scientists were often a "jack of all trades" with a mix of skills including statistical analysis and large-scale processing, many organizations eventually differentiated the roles of data scientist (more focused on analytical tasks) and data engineer (more focused on analysis infrastructure). Moreover, traditional IT departments still play a critical role in organization-wide data efforts, but were not a focus of our interview study.

In the six years since our interviews, engagement with customers (including user studies, site visits, surveys, and interviews) has led us to refine and expand our personas. For example, while highly-skilled data scientists remain a constituency, we focus on them less than we did on campus. Users with programming ability often choose to complete data wrangling tasks using code, foregoing the efficiency and feedback afforded by improved visual tools in favor of the familiar walled garden of their favorite programming language. On the other hand, analysts in a specific line of business have driving questions and deep domain expertise, but may lack either the time or programming knowledge to write scalable data transformation code. These users also greatly outnumber data scientists, and so represent a larger business opportunity. Separately, we have learned to appreciate an additional class of user other than the data analyst: users tasked with operationalizing, scheduling, and deploying

workflows generated by self-service analysts. These users often need to oversee the use of computing resources, and the governance of data—an issue made all the more clear by the recent focus on privacy and the GDPR legislation in Europe. Some Trifacta customers choose the product not for its ease of use, but because it can be used to ensure a reliable, auditable process for preparing data for legislative or administrative compliance.

3.2 Interface Evolution

Data wrangling is fundamentally a programming task: users must choose a sequence of transformations to raw data, mapping it into a more usable and actionable form. Many data preparation solutions, including ours, formulate a *domain-specific language (DSL)* for expressing the various transformations needed, such as formatting, filtering, aggregation, and integration operations. While writing high-level DSL programs can be an attractive alternative to writing low-level code in a more general language, it still imposes a technical burden. Instead, at Trifacta we enable end users to express DSL statements through interactions with a graphical user interface.

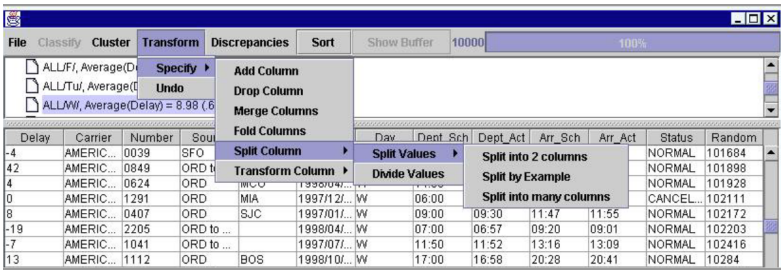


Figure 1: Potter's Wheel with its menu-driven interface

A primary source of inspiration was Raman & Hellerstein's Potter's Wheel system [29]. Potter's Wheel involved a scalable table grid view for directly inspecting data, coupled with menu-driven commands (and accompanying modal dialogs) for applying transformations (Figure 1). These graphical commands were translated into statements in an underlying DSL: a sequence of user commands generates a data transformation script.

In lieu of menus, our Data Wrangler research prototypes [18, 10] focused on more direct manipulation interactions. We initially explored interface gestures for expressing transformations, but this led to ambiguity, as the same gesture might be used for different transformations. For example, if you select a span of text, do you intend to extract, replace, or split up the text? However, we realized that for a range of simple interactions (e.g., row, column, and text selection), typically only a small number of transformations make sense. This insight led us to an approach called *predictive interaction*, analogous to auto-complete, in which simple gestures guide automated predictions of which transformations to apply. The system searches over relevant DSL statements compatible with the user selection and recommends those it believes will most improve the data [11]. To help users select among the suggestions, we provided transformation summaries in a natural text format and visualizations that conveyed the effect the transformation would have on the data table (Figure 2).

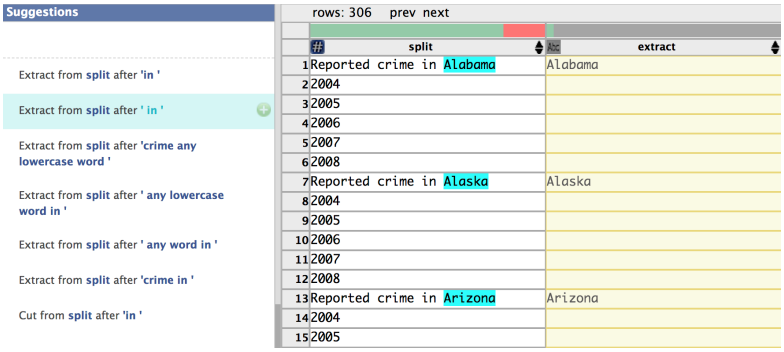


Figure 2: Wrangler research prototype, with predictive interaction.

After founding Trifacta, we rebuilt the DSL and interface from the ground up. We maintained the predictive interaction approach, including visual transform previews. We incorporated more powerful means of transform customization. For example, predictions may be close to a user's intent but not a perfect match, requiring further refinement. Users also needed means to express custom calculation formulae. These requirements, combined with a startup's zeal for agility, led to us to an early version that surfaced recommendations directly as DSL code

statements, alongside a code editor for drafting or modifying statements (Figure 3). This provided maximal expressiveness, but required learning the syntax of our Wrangle DSL. While valuable for trained users, user feedback indicated that this design imposed an unattractive learning curve for our target personas.

We also incorporated *visual profiles*: automatically-generated summary visualizations that convey the shape and structure of the data. We added histograms to the column headers in the grid view, while also providing dedicated per-column views with statistical summaries and type-specific visualizations, such as maps for geospatial data and aggregate trends by month, week, day, *etc.* for temporal data. Akin to interaction with table cells, interaction with visual profiles trigger transform suggestions, for example filtering to a selected data range.

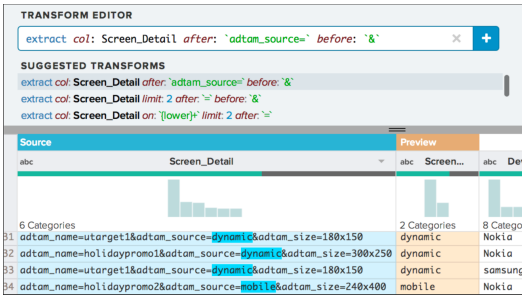


Figure 3: An early version of Trifacta Wrangler, with explicit DSL editing.

As both our product and personas evolved, we further elaborated the interface model with additional features. For example, we sought to improve users' ability to interpret transform suggestions through the inclusion of a navigable gallery of small visual summaries of suggested transformations. As shown in Figure 4, these *cards* pair DSL code snippets with before & after comparisons to complement our existing visual previews.

To help users create new transforms or refine suggested ones without writing textual code, we developed a transform *builder*: a structured graphical editor for transformation authoring and modification. Builder scaffolds the transformation authoring process with input widgets for each transform parameter (Figure 5). Users can directly configure a transform without having to learn the syntax of the underlying DSL. Builder also provides *in situ* help and documentation to further aid the process. The transform types and parameters in builder are isomorphic to those in Wrangle DSL, facilitating learning over time. Finally, we found that many users migrating from traditional tools like Excel wanted a menu bar to browse for functionality; choosing an item from Trifacta's menu bar populates the builder with suggested parameters.

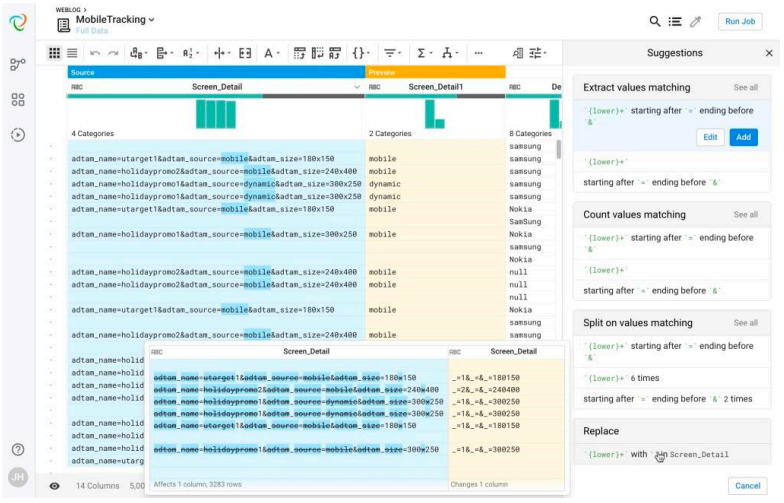


Figure 4: Trifacta Wrangler with suggestion cards. The first suggestion highlighted in blue is previewed in the main grid. Hovering over an alternative suggestion (low in the list) yields a visual preview of its effects.

3.3 Architecture Evolution

The original Potter's Wheel tool was built as a desktop application, centered on a DSL of transforms that were surfaced as menu commands [29]. (Many current data preparation tools are designed this way.) Potter's Wheel was distinguished architecturally by a unique ability to allow instantaneous interaction on large datasets. It would begin scanning a file or table, and stream it immediately into an online reordering algorithm that bucketized the incoming data [31]. Sample rows would immediately appear in the user interface from buckets biased toward the current scrollbar position, giving the illusion of fluid scrolling over an unbounded table of data. Meanwhile,

streaming anomaly detection algorithms could work in the background on the data while it was being scanned.

Ten years later, the academic Wrangler prototype [18] was implemented as a browser-based Javascript application. It accepted only very small datasets pasted into a text buffer, but focused on generating not only output data, but output *scripts* in one of a variety of external languages including Javascript, Python, SQL and MapReduce. This shift in primary focus from data manipulation to code generation was critical in the design of Trifacta.

When designing Trifacta's first version in 2012, we were interested in extending the reach of Wrangler beyond casual web use to a full range of challenging data-rich scenarios. Our goals included tackling a rich variety of data representations, from unstructured to semi-structured and relational data. We were also determined to address potentially unbounded volumes of data—not just generating "big data" code as a proof of concept, but working in a seamless and performant way in a big data ecosystem.

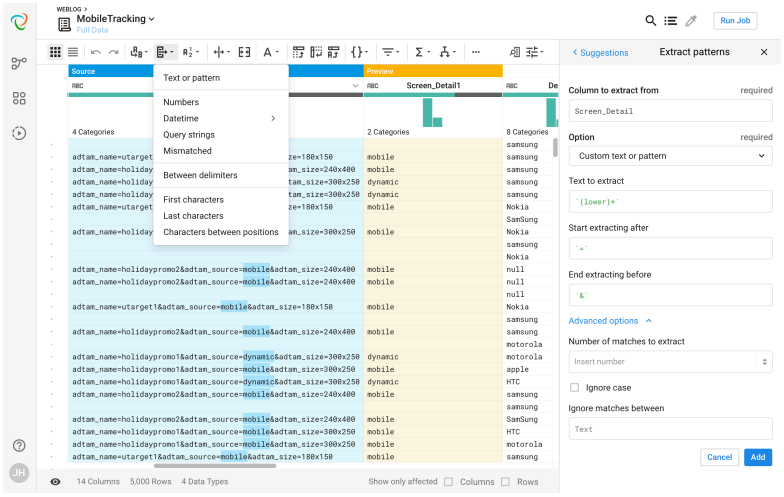


Figure 5: Menus and builder for explicit transformation. Choosing a menu item opens and populates the builder interface on the right, which provides a structured graphical editor over the DSL code from Figure 3.

This led us to consider two deployment targets. The first was to host Trifacta Wrangler, this time as a scalable service embedded in public cloud infrastructure. The second choice was to deliver an on-premises solution that would integrate with the emerging "big data" stack. Architecturally, we believed that a design targeted to run as a service would also work well on premises, so we essentially designed for running as a service. As it happened, 2012 was far too early for most enterprise customers to bring serious data to the cloud, so we opted to deliver Trifacta as an on-premises "big data" solution first. However, when Google approached us four years later to deliver Trifacta as a service called Google Cloud Dataprep, we were architecturally in a good position to meet that challenge, and now we have offerings on all three public clouds.

3.3.1 Trifacta Architecture

The key goals of Trifacta's coarse-grained architecture were to scale from a single-user desktop up to the biggest SaaS deployments in the cloud, while preserving end-user simplicity and immediacy at every scale.

Trifacta was designed from the start as a three-tier solution, consisting of a Trifacta client built on browser technology (currently both Javascript and WebAssembly), a middle tier of soft-state Trifacta services, and a lower tier of commodity (non-Trifacta) services for persistence and big-data compute. The lower tier was designed to be pluggable with a variety of cloud-hosted or on-premises solutions including the Apache big data stack for on-premises deployment, and the native services of the popular public clouds. Key to this design was the focus going back to Potter's Wheel of having a DSL representation of user intent at the heart of the system, and—like Wrangler—an ability to cross-compile that DSL to multiple candidate execution services.

The top ("client") tier of Trifacta runs in a web browser, or in a desktop application using web components via the Chromium framework. The Trifacta client combines an intelligent Transformer user interface with a DSL cross-compiler—both written in Javascript—and also includes a C++-based in-memory query engine called Photon [12] that is compiled to WebAssembly to run in the browser. The user interface includes many of Trifacta's characteristic visual profiling and ML-based predictive interaction [11] model for transformation,

as well as the ability to manage flows and external datasets from a wide variety of remote sources: filesystems, web-based storage, databases and so on. When a user identifies a dataset of interest, Trifacta's middle tier fetches the raw data and pushes it into the browser for initial visualization; if the dataset will not fit in browser memory, the middle tier pushes a sample of the data into the browser. As the user manipulates data in the Transformer interface, "recipes" (transformation scripts) are generated in Trifacta's DSL, "Wrangle". For every action in the Transformer interface, the recipe up to that point is compiled into a Photon query plan, executed by Photon on the browser-based data, and rendered on screen in a range of visualizations. This provides a rich, responsive direct-manipulation experience in the browser, and ensures that users can fluidly explore and transform their data without any backend processing. When the user chooses to run or schedule a job, the front-end cross-compiles their recipe to the backend execution framework of choice in their deployment—the list of targets has evolved over time as big data engines have waxed and waned in popularity, but current offerings at this date include hosted editions of Photon, Apache Spark, Google Dataflow, AWS Elastic MapReduce, and Azure HDInsight.

Trifacta's rich interface places significant demands on Photon in the browser: nearly every user gesture generates a multitude of transformation tasks (queries)—different tasks for each different suggestion to the user—which all need to be run through Photon at the speed of human interaction to generate visual previews. This need for concurrent, responsive query processing is the primary reason we built our own in-memory query engine. Some products for data transformation use backend technologies like Spark for their interactive processing. In designing Photon, we evaluated Spark to back each client, but found that Spark's performance was far too unpredictable—in startup, latency and especially memory pressure—to service our interactive user experience. Instead, Photon was inspired by a number of similar high-performance in-memory query engines including Hyper [20] (now embedded similarly in Tableau) and Cloudera Impala [21]. An overview of the Photon architecture is beyond the scope of this paper, but was presented in a conference talk viewable online [12].

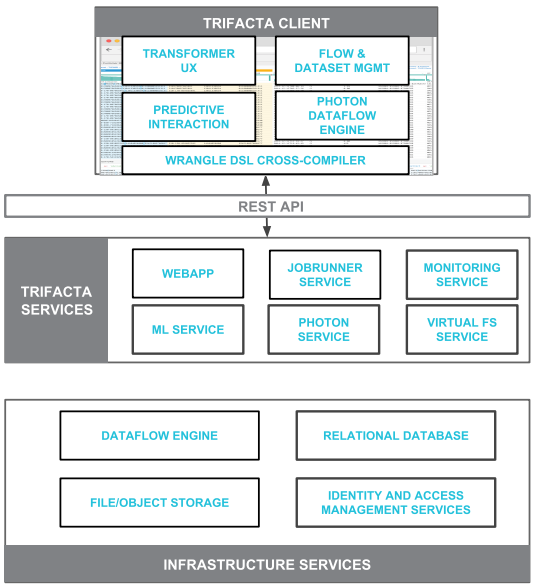


Figure 6: Sketch of Trifacta Wrangler Enterprise.

Trifacta's middle tier is a set of soft-state services that power the frontend, and connect it to scalable data processing and storage. First, there is a web application that routes requests and controls the user interface's metadata, including browsing and storage of a registry of user Flows, Datasets, scheduled jobs, collaborations with other users, and so on. There is a jobrunner for running transformation jobs in big data engines in the lower tier, and a monitoring service to track those jobs throug their lifecycle. There is a "virtual filesystem" service for abstracting access to datasets from various storage systems, and a backend C++ Photon engine for running jobs that are too large to run in the browser, but small enough to run most quickly in memory on a single machine. Finally there is a machine learning (ML) service for serving predictions from a variety of models that power Trifacta's intelligent features. A non-exhaustive list of these ML features includes auto-detection of data types, recommendation of transformation steps, recommendations for join keys, synthesis of regular expressions from textual examples, and extraction/conversion of canonical patterns for variant data (e.g., for understanding a mixture of date formats, log records, addresses and so on, and optionally standardizing them.)

The bottom tier of a Trifacta installation is not Trifacta code at all—it is a set of commodity infrastructure services that provide persistence and compute resources. These include a relational database for maintaining user metadata for the webapp, a scalable file system for unstructured dataset inputs and outputs, a dataflow execution service (query processor) and native services for identity and access management (e.g., Kerberos).

Trifacta also offers a REST API for integrating with third-party services and applications: for example for publishing clean datasets to downstream analytics or charting tools, or for reading and writing metadata and job lineage to third party data catalogs. This API is critical for Trifacta to "play nice" with other pieces of the data stack, including infrastructure as well as downstream user-facing applications.

3.3.2 Agile Exploration of Big Data: Sampling

A key design imperative for Trifacta is to allow users to see and directly manipulate their data during transformation, which requires a responsive user interface. For very large datasets, the only way to get responsive interaction is to reduce the data to an efficient size—either via precomputation (summary structures like data cubes) or via subsetting the data (using indexes or random samples). Because the process of data wrangling transforms the dataset step-by-step, it is infeasible to assume precomputed structure like cubes or indexes are available on the full dataset—these structures would have to be rebuilt after every transform step! Instead, for data wrangling to both scale up and provide an immediate, responsive user experience, sampling is in some form is quite natural. Data preparation products that do not offer built-in sampling either (a) do not provide a responsive user experience, or (b) effectively force the user to do sampling by hand outside the product. Worse, when users do their own sampling in many products, they often are only able to get a transformed copy of their sample data—the wrangling work they author in the product is not transferrable to the larger dataset.

Potter's Wheel scaled fluidly via a unique sampling technique integrated into the UI. The design was innovative but not without problems: the sampling was a best-effort approach on streaming read, and had randomness guarantees that were hard to interpret statistically. The user experience was unpredictable as well, in terms of what data would actually be seen during scrolling: in particular, if a user saw a particular row at some scrollbar location, it was not guaranteed that said row would still be there if they scrolled away and scrolled back [30].

Wrangler, by contrast, took a simple approach that put the burden on the user. Because Wrangler could only consume as much data as the OS "clipboard" could cut-and-paste, users with big datasets had to downsample their data themselves. Importantly, though, Wrangler could still be useful for users with big data sets: after the user transformed their sample dataset to their satisfaction, Wrangler (like Potter's Wheel before it) could output a reusable transformation script to transform the full dataset using a variety of programming languages (Javascript, Python, SQL, etc.) Hence while Wrangler itself did not scale, it provided a clear "contract" to the user about how it could and could not help with large data sets.

Trifacta was designed to preserve Wrangler's spirit of user clarity, but with significantly better scale and automation using an approach we call *sample-to-scale*. The goals of sample-to-scale are to guarantee a responsive but rich visual experience, to put users in control of sampling in intuitive and low-friction ways, and to ensure that users' visual work is encoded in recipes that can be run at scale on unbounded amounts of data. To get started quickly when opening a new dataset, Trifacta loads the "head" of the dataset into the browser by default, while in the background Trifacta's services are assembling a statistically robust simple random sample on the fly. The user is alerted when the random sample is available, and can switch between samples in the browser at any time. Because Trifacta user behaviors are captured in declarative recipes, the new sample can be run through the existing recipe and rendered in the browser immediately, to allow the user to see how their recipe to that point would work on a different sample. Trifacta also allows users to specify specific sampling methods including stratified sampling on particular columns, samples that ensure that certain values appear in their output, or samples that ensure that certain anomalies of interest are in evidence so they can be addressed. For UX consistency, Trifacta remembers the samples that users have worked with, and ensures that it is possible to go back to prior states of transformation history with the same samples that were originally seen. Finally, when users are satisfied with the properties of their transformed sample (or samples) they can run a full job, which executes the recipe over the entire datasets and produces both an output file and a data profile.

3.4 Deployments

Trifacta's architecture allowed us to deploy it in multiple environments. There are three basic patterns.

In an **on-premises** deployment, the lowest tier is the customer's existing installation of Hadoop and Spark from vendors like Cloudera and Hortonworks. Trifacta's middle tier is deployed on an "edge node" that sits alongside the Hadoop cluster, and provides metadata storage via a relational database.

We also offer a similar **cloud tenant** deployment for customers whose data lives in the cloud. In this format, Trifacta's middle tier runs in Kubernetes-managed containers on behalf of the client, and native cloud services (database, dataflow, filesystem) replace the metadata database and the various functions of Hadoop.

Trifacta can also be deployed as **multitenant SaaS**, as we do for Google Cloud Dataprep. In this configuration, Trifacta's middle tier is run in an elastic environment of containers managed with Kubernetes. Jobs are compiled to run on elastic cloud dataflow services like Google Cloud Dataflow. Metadata is stored in a multitenant cloud database. We have seen users of this deployment routinely transform petabytes of data and consume thousands of compute hours. Yet they share the same agile experience in the browser enjoyed by smaller user cases thanks to the immediacy of photon and the philosophy of sample-to-scale.

To stay connected to our roots in the academic Wrangler prototype, we maintain a free photon-based edition of Trifacta Wrangler for use by the general public at `https://www.trifacta.com`. The global community of data wranglers provides us with front-line awareness of how the software can be improved over time.

4 Conclusion

After years of scattered research, Self-Service Data Preparation has emerged as a significant category in the data industry. It has a different technical focus than legacy data integration/ETL due to evolution in the users who wrangle data, the variety of data they face, and the data systems they favor. Usage at major enterprises demonstrates that new technologies improve processes for experienced users, and enable new classes of users to wrangle their own data. The area remains important for further innovation across Databases, AI, PL and HCI; we encourage our fellow researchers to further explore industry reports on the area [1, 5, 8, 23, 35] as well as free offerings of data preparation solutions online to get a sense of issues in the field.

References

[1] Bloor Research International. Self-Service Data Preparation and Cataloguing, 2016. `https://www.bloorresearch.com/research/self-service-data-preparation-cataloguing/` accessed 05/14/2018.

[2] Center for Disease Control. The anatomy of an HIV outbreak response in a rural community, 2015. `https://blogs.cdc.gov/publichealthmatters/2015/06/the-anatomy-of-an-hiv-outbreak-response-in-a-rural-community/`, accessed 5/11/2018.

[3] Xu Chu, et al. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2201–2206. ACM, 2016.

[4] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration*. Elsevier, 2012.

[5] Dresner Advisory Services. End User Data Preparation Study, 2015. `https://www.trifacta.com/gated-form/trifacta-earns-top-ranking-in-end-user-data-preparation-study/`, accessed 05/14/2018.

[6] Kim Flaherty. Why personas fail, 2018. `https://www.nngroup.com/articles/why-personas-fail/`, accessed 5/11/2018.

[7] Helena Galhardas, et al. Ajax: an extensible data cleaning tool. In *ACM Sigmod Record*, 29(2):590, ACM, 2000.

[8] Gartner. Peer Insights: Data Preparation. `https://www.gartner.com/reviews/market/data-preparation-tools`, accessed 05/14/2018.

[9] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN Notices*, volume 46, pages 317–330. ACM, 2011.

[10] Philip J. Guo, et al. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proc. ACM User Interface Software & Technology (UIST)*, 2011.

[11] Jeffrey Heer, Joseph Hellerstein, and Sean Kandel. Predictive interaction for data transformation. In *Proc. Conference on Innovative Data Systems Research (CIDR)*, 2015.

[12] Joseph M. Hellerstein and Seshadri Mahalingam. Architecting Immediacy—The Design of a High-Performance Portable Wrangling Engine. In *Strata Conference*, March 31, 2016.

[13] Mauricio A Hernández, Renée J Miller, and Laura M Haas. Clio: A semi-automatic tool for schema mapping. *ACM SIGMOD Record*, 30(2):607, 2001.

[14] Ravi Hubbly, 2018. `https://www.trifacta.com/partners/leidos/`, accessed 05/16/2018.

[15] Ihab F Ilyas and Xu Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 5(4):281–393, 2015.

[16] Zhongjun Jin, et al. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 683–698. ACM, 2017.

[17] Sean Kandel, et al. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011.

[18] Sean Kandel, et al. Wrangler: Interactive visual specification of data transformation scripts. In *Proc. ACM Human Factors in Computing Systems (CHI)*, 2011.

[19] Sean Kandel, et al. Enterprise data analysis and visualization: An interview study. In *Proc. IEEE Visual Analytics Science & Technology (VAST)*, 2012.

[20] Alfons Kemper and Thomas Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE*, pages 195–206, 2011.

[21] Marcel Kornacker and Justin Erickson. Cloudera impala: Real time queries in apache hadoop, for real. *Cloudera Engineering Blog*, October 2012.

[22] Leidos. Deploying new tools for the fight against disease outbreaks, 2016. `https://www.leidos.com/sites/default/files/responsive/Health/CAADS-Case-Study.pdf`, accessed 5/11/2018.

[23] Cinny Little, Gene Leganza, and Jun Lee. *The Forrester Wave: Data Preparation Tools, Q1 2017*. Forrester, 2017.

[24] Yitzhak Mandelbaum, et al. PADS/ML: A functional data description language. *ACM SIGPLAN Notices*, 42(1):77–83, ACM, 2007.

[25] Steve Lohr. For Big-Data Scientists, Janitor Work Is Key Hurdle to Insights. *The New York Times*, August 17, 2014.

[26] Adam Marcus and Aditya Parameswaran. Crowdsourced data management: Industry and academic perspectives. *Foundations and Trends® in Databases*, 6(1-2):1–161, 2015.

[27] Jordan Novet. Google launches Cloud Dataprep, an embedded version of Trifacta. *VentureBeat*, March 9, 2017.

[28] DJ Patil. *Data Jujitsu*. O'Reilly Media, Inc., 2012.

[29] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[30] Vijayshankar Raman and Joseph M. Hellerstein. Partial results for online query processing. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 275–286. ACM, 2002.

[31] Vijayshankar Raman, Bhaskaran Raman, and Joseph M. Hellerstein. Online dynamic reordering for interactive data processing. In *VLDB*, volume 99, pages 709–720, 1999.

[32] T Rattenbury, et al. *Data Wrangling: Techniques and Concepts for Agile Analytics*. O'Reilly, Sebastopol, CA, 2015.

[33] Megan Twohey. Mike Pence's response to H.I.V. outbreak: Prayer, then a change of heart. *The New York Times*, August 7, 2016.

[34] Alex Woodie. Tracking the opioid-fueled outbreak with big data. *Datanami*, December 15, 2016.

[35] Ehtisham Zaidi, Rita L. Sallam, and Shubhangi Vashisth. *Market Guide for Data Preparation*. Gartner, December 14, 2017.

Toward a System Building Agenda for Data Integration (and Data Science)

AnHai Doan, Pradap Konda, Paul Suganthan G.C., Adel Ardalan, Jeffrey R. Ballard, Sanjib Das, Yash Govind, Han Li, Philip Martinkus, Sidharth Mudgal, Erik Paulson, Haojun Zhang
University of Wisconsin-Madison

Abstract

We argue that the data integration (DI) community should devote far more effort to building systems, in order to truly advance the field. We discuss the limitations of current DI systems, and point out that there is already an existing popular DI "system" out there, which is PyData, the open-source ecosystem of 138,000+ interoperable Python packages. We argue that rather than building isolated monolithic DI systems, we should consider extending this PyData "system", by developing more Python packages that solve DI problems for the users of PyData. We discuss how extending PyData enables us to pursue an integrated agenda of research, system development, education, and outreach in DI, which in turn can position our community to become a key player in data science. Finally, we discuss ongoing work at Wisconsin, which suggests that this agenda is highly promising and raises many interesting challenges.

1 Introduction

In this paper we focus on data integration (DI), broadly interpreted as covering all major data preparation steps such as data extraction, exploration, profiling, cleaning, matching, and merging [10]. This topic is also known as data wrangling, munging, curation, unification, fusion, preparation, and more. Over the past few decades, DI has received much attention (e.g., [37, 29, 31, 20, 34, 33, 6, 17, 39, 22, 23, 5, 8, 36, 15, 35, 4, 25, 38, 26, 32, 19, 2, 12, 5, 16, 2, 3]). Today, as data science grows, DI is receiving even more attention. This is because many data science applications must first perform DI to combine the raw data from multiple sources, before analysis can be carried out to extract insights.

Yet despite all this attention, today we do not really know whether the field is making good progress. The vast majority of DI works (with the exception of efforts such as Tamr and Trifacta [36, 15]) have focused on developing *algorithmic solutions*. But we know very little about whether these (ever-more-complex) algorithms are indeed useful in practice. The field has also built mostly isolated *system prototypes*, which are hard to use and combine, and are often not powerful enough for real-world applications. This makes it difficult to decide what to *teach* in DI classes. Teaching complex DI algorithms and asking students to do projects using our prototype systems can train them well for doing DI research, but are not likely to train them well for solving real-world DI problems in later jobs. Similarly, *outreach to real users* (e.g., domain scientists) is difficult. Given that we have mostly focused on "point DI problems", we do not know how to help them solve *end-to-end* DI tasks. That is,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

we cannot tell them how to start, what algorithms to consider, what systems to use, and what they need to do manually in each step of the DI process.

In short, today our DI effort in research, system development, education, and outreach seem disjointed from one another, and disconnected from real-world applications. As data science grows, this state of affairs makes it hard to figure out how we can best relate and contribute to this major new field.

In this paper we take the first steps in addressing these problems. We begin by arguing that the key to move forward (and indeed, to tie everything together) is to devote far more effort to *building DI systems*. DI is engineering by nature. We cannot just keep developing DI algorithmic solutions in a vacuum. At some point we need to build systems and work with real users to evaluate these algorithms, to integrate disparate R&D efforts, and to make practical impacts. In this aspect, DI can take inspiration from RDBMSs and Big Data systems. Pioneering systems such as System R, Ingres, Hadoop, and Spark have really helped push these fields forward, by helping to evaluate research ideas, providing an architectural blueprint for the entire community to focus on, facilitating more advanced systems, and making widespread real-world impacts.

We then discuss the limitations of current DI systems, and point out that there is already an existing DI system out there, which is very popular and growing rapidly. This "system" is PyData, the open-source ecosystem of 138,000+ interoperable Python packages such as pandas, matplotlib, scikit-learn, etc. We argue that rather than building isolated monolithic DI systems, *the DI community should consider extending this PyData "system", by developing Python packages that can interoperate and be easily combined to solve DI problems for the users of PyData*. This can address the limitations of the current DI systems, provide a system for the entire DI community to rally around, and in general bring numerous benefits and maximize our impacts.

We propose to extend the above PyData "system" in three ways:

- For each end-to-end DI scenario (e.g., entity matching with a desired F_1 accuracy), develop a "how-to guide" that tells a power user (i.e., someone who can program) how to execute the DI process step by step, identify the true "pain points" in this process, develop algorithmic solutions for these pain points, then implement the solutions as Python packages.

- Foster PyDI, an ecosystem of such DI packages as a part of PyData, focusing on how to incentivize the community to grow PyDI, how to make these packages seamlessly interoperate, and how to combine them to solve larger DI problems.

- Extend PyDI to the cloud, collaborative (including crowdsourcing), and lay user settings.

We discuss how extending PyData can enable our community to pursue an *integrated* agenda of research, system development, education, and outreach. In this agenda we develop solutions for real-world problems that arise from solving end-to-end DI scenarios, build real-world tools into PyData, then work with students and real-world users on using these tools to solve DI problems. We discuss how this agenda can position our community to become a key player in data science, who "owns" the data quality part of this new field. Finally, we describe initial work on this agenda in the past four years at Wisconsin. Our experience suggests that this agenda is highly promising and raises numerous interesting challenges in research, systems, education, and outreach.

2 Data Science and Data Integration

In this section we briefly discuss data science, data integration, and the relationship between the two.

Currently there is no consensus definition for data science (DS). For our purposes, we will define data science as a field that develops principles, algorithms, tools, and best practices to manage data, focusing on three topics: (a) analyzing raw data to infer insights, (b) building data-intensive artifacts (e.g., recommender systems, knowledge bases), and (c) designing data-intensive experiments to answer questions (e.g., A/B testing). As such, DS is clearly here to stay (even though the name may change), for the simple reason that everything is now data driven, and will only become even more so in the future.

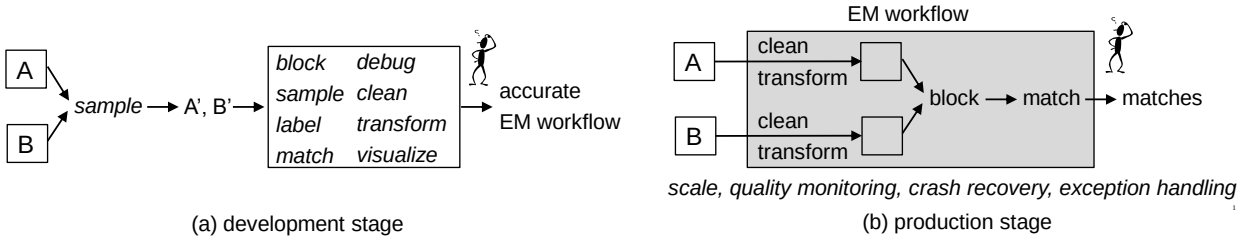


Figure 1: Matching two tables in practice often involves two stages and many steps (shown in italics).

In this paper we focus on the first topic, analyzing raw data to infer insights, which has received a lot of attention. A DS task here typically consists of two stages. In the *data integration (DI) stage*, raw data from many different sources is acquired and combined into a single clean integrated dataset. Then in the *data analysis stage*, analysis is performed on this dataset to obtain insights. Both stages extensively use techniques such as visualization, learning, crowdsourcing, Big Data scaling, and statistics, among others.

Core DI Problems & End-to-End DI Scenarios: The DI stage is also known as data wrangling, preparation, curation, cleaning, munging, etc. Major problems of this stage include extraction, exploration, profiling, cleaning, transforming, schema matching, entity matching, merging, etc. We refer to these as *core DI problems*. When solving a core DI problem, real-world users often want to reach a desired outcome (e.g., at least 95% precision and 80% recall). We refer to such desired outcomes as *goals*, and to such scenarios, which go from the raw data to a goal, as *end-to-end DI scenarios*. (As a counter example, simply trying to maximize the F_1 accuracy of entity matching, as many current research works do, is not an end-to-end scenario.)

Development and Production Stages: To solve an end-to-end DI scenario, a user typically goes through two stages. In the *development stage*, the user experiments to find a DI workflow that can go from raw data to the desired goal. This is often done using data samples. In this stage, the user often has to explore to understand the problem definition, data, and tools, and make changes to them if necessary. Then in the *production stage*, the user specifies the discovered workflow (e.g., declaratively or using a GUI), optionally optimizes, then executes the workflow on the entirety of data. (Sometimes the steps of these two stages may be interleaved.)

Example 1: *Consider matching two tables A and B each having 1M tuples, i.e., find all pairs $(a \in A, b \in B)$ that refer to the same real-world entity. In the development stage (Figure 1.a), user U tries to find an accurate EM workflow. This is often done using data samples. Specifically, U first samples two smaller tables A' and B' (each having 100K tuples, say) from A and B. Next, U performs blocking on A' and B' to remove obviously non-matched tuple pairs. U often must try and debug different blocking techniques to find the best one.*

Suppose U wants to apply supervised learning to match the tuple pairs that survive the blocking step. Then next, U may take a sample S from the set of such pairs, label pairs in S (as matched / non-matched), and then use the labeled sample to develop a learning-based matcher (e.g., a classifier). U often must try and debug different learning techniques to develop the best matcher. Once U is satisfied with the accuracy of the matcher, the production stage begins (Figure 1.b). In this stage, U executes the EM workflow that consists of the developed blocking strategy followed by the matcher on the original tables A and B. To scale, U may need to rewrite the code for blocking and matching to use Hadoop or Spark.

3 Limitations of Current Data Integration Systems

Each current DI system tries to solve either *a single core DI problem* or *multiple core DI problems jointly* (e.g., schema matching, followed by schema integration, then EM). We now discuss these two groups in turn. Consider systems for a single core DI problem. Our experience suggests that this group suffers from the following limitations.

1. Do Not Solve All Stages of the End-to-End DI Process: Most current DI systems support only the production stage. For example, most current EM systems provide a set of blockers and matchers. The user can specify an EM workflow using these blockers/matchers (either declaratively or via a GUI). The systems then

optimize and execute the EM workflow. Much effort has been devoted to developing effective blocker/matcher operators (e.g., maximizing accuracy, minimizing runtime, minimizing crowdsourcing cost, etc.). There has been relatively little work on the development stage. It is possible that database researchers have focused mostly on the production stage because it follows the familiar query processing paradigm of RDBMSs. Regardless, we cannot build practical DI systems unless we also solve the development stage.

2. Provide No How-To Guides for Users: Solving the development stage is highly non-trivial. There are three main approaches. First, we can try to completely automate it. This is unrealistic. Second, we can still try to automate, but allowing limited human feedback at various points. This approach is also unlikely to work. The main reason is that the development stage is often very messy, requiring multiple iterations involving many subjective judgments from the human user. Very often, after working in this stage for a while, the user gains a better understanding of the problem and the data at hand, then revises many decisions on the fly.

Example 2: *Consider the labeling step in Example 1. Labeling tuple pairs in sample S as matched or non-matched seems trivial. Yet it is actually quite complicated in practice. Very often, during the labeling process user U gradually realizes that his/her current match definition is incorrect or inadequate. For instance, when matching restaurant descriptions, U may start with the definition that two restaurants match if their names and street addresses match. But after a while, U realizes that the data contains many restaurants that are branches of the same chain (e.g., KFC). After checking with the business team, U decides that these should match too, even though their street addresses do not match. Revising the match definition however requires U to revisit and potentially relabel pairs that have already been labeled, a tedious and time-consuming process.*

As another example, suppose a user U wants to perform EM with at least 95% precision and 80% recall. How should U start? Should U use a learning-based or a rule-based EM approach? What should U do if after many tries U still cannot reach 80% recall with a learning-based approach? It is unlikely that an automated approach with limited human feedback would work for this scenario.

As a result, it is difficult to imagine that the development stage can be automated with any reasonable degree soon. In fact, today it is still often executed using the third approach, where a human user drives the end-to-end process, making decisions and using (semi-)automated tools in an *ad-hoc* fashion. Given this situation, many users have indicated to us that what they really need, first and foremost, is *a how-to guide* on how to execute the development stage. Such a guide is *not* a user manual on how to use a tool. Rather, it is a detailed step-by-step instruction to the user on how to start, when to use which tools, and when to do what manually. Put differently, *it is an (often complex) algorithm for the human user to follow*. Current DI systems lack such how-to guides.

3. Provide Few Tools for the Pain Points: When executing the development state, a user often runs into many "pain points" and wants (semi-)automated tools to solve them. But current DI systems have provided few such tools. Some pain points are well known, e.g., debugging blockers/matchers in EM. Many more are not well known today. For example, many issues thought trivial turn out to be major pain points in practice.

Example 3: *Exploring a large table by browsing around is a major pain point, for which there is no effective tool today (most users still use Excel, OpenRefine, or some limited browsing capabilities in PyData). Counting the missing values of a column, which seems trivial, turns out to be another major pain point. This is because in practice, missing values are often indicated by a wide range of strings, e.g., "null", "none", "N/A", "unk", "unknown", "-1", "999", etc. So the user often must painstakingly detect and normalize all these synonyms, before being able to count. Labeling tuple pairs as match/no-match in EM is another major pain point. Before labeling, users often want to run a tool that processes the tuple pairs and highlights possible match definitions, so that they can develop the most comprehensive match definition. Then during the labeling process, if users must still revise the match definition, they want a tool that quickly flags already-labeled pairs that may need to be relabeled.*

4. Difficult to Exploit a Wide Variety of Capabilities: It turns out that even when we just try to solve a single core DI problem, we already have to utilize a wide variety of capabilities. For example, when doing EM, we

often have to utilize SQL querying, keyword search, learning, visualization, crowdsourcing, etc. Interestingly, we also often have to solve *other DI problems*, such as exploration, cleaning, information extraction, etc. So we need the solution capabilities for those problems as well.

Today, it is very difficult to exploit all these capabilities. Incorporating all of them into a single DI system is difficult, if not impossible. The alternative solution of moving data among multiple systems, e.g., an EM system, an extraction system, a visualization system, etc., also does not work. This is because solving a DI problem is often an iterative process. So we would end up moving data among multiple systems *repeatedly*, often by reading/writing to disk and translating among proprietary data formats numerous times, in a tedious and time consuming process. A fundamental problem here is that most current DI systems are *stand-alone monoliths* that are not designed to interoperate with other systems. Put differently, most current DI researchers are still building stand-alone systems, rather than developing an ecosystem of interoperable DI systems.

5. Difficult to Customize, Extend, and Patch: In practice, users often want to customize a generic DI system to a particular domain. Users also often want to extend the system with latest technical advances, e.g., crowdsourcing, deep learning. Finally, users often have to write code, e.g., to implement a lacking functionality or combine system components. Writing "patching" code correctly in "one shot" (i.e., one iteration) is difficult. Hence, ideally such coding should be done in an interactive scripting environment, to enable rapid prototyping and iteration. Few if any of the current DI systems are designed from scratch such that users can easily customize, extend, and patch in many flexible ways. Most systems provide "hooks" at only certain points in the DI pipeline for adding limited new functionalities (e.g., a new blocker/matcher), and the vast majority of systems are not situated in an interactive scripting environment, making patching difficult.

6. Similar Problems for the Production Stage: So far we have mostly discussed problems with the development stage. But it appears that many of these problems may show up in the production stage too. Consider for example a domain scientist U trying to execute an EM workflow in the production stage on a single desktop machine and it runs too slowly. What should U do next? Should U try a machine with bigger memory or disk? Should U try to make sure the code indeed runs on multiple cores? Should U try some of the latest scaling techniques such as Dask ($dask.pydata.org$), or switch to Hadoop or Spark? Today there is no guidance to such users on how best to scale a DI workflow in production.

Limitations of Systems for Multiple DI Problems: So far we have discussed systems for a single core DI problem. We now discuss systems for multiple DI problems. Such a system jointly solves a set of DI problems, e.g., data cleaning, schema matching and integration, then EM. This helps users solve the DI application seamlessly end-to-end (without having to switch among multiple systems), and enables runtime/accuracy optimization across tasks. Our experience suggests that these systems suffer from the following limitations. (1) For each component DI problem, these systems have the same problems as the systems for a single DI problems. (2) As should be clear by now, building a system to solve a single core DI problem is already very complex. Trying to solve multiple such problems (and accounting for the interactions among them) in the same system often exponentially magnifies the complexity. (3) To manage this complexity, the solution for each component problem is often "watered down", e.g., fewer tools are provided for both the development and production stages. This in turn makes the system less useful in practice. (4) If users want to solve just 1-2 DI problems, they still need to install and load the entire system, a cumbersome process. (5) In many cases optimization across problems (during production) does not work, because users want to execute the problems one by one and materialize their outputs on disk for quality monitoring and crash recovery. (6) Finally, such systems often handle only a pre-specified set of workflows that involves DI problems from a pre-specified set. If users want to try a different workflow or need to handle an extra DI problem, they need another system, and so end up combining multiple DI systems anyway.

4 The PyData Ecosystem of Open-Source Data Science Tools

In the past decade, using open-source tools to do data science (for both data integration and data analysis stages) has received significant growing attention. The two most well-known ecosystems of such open-source tools are in Python and R. In this paper we focus on the Python ecosystem, popularly known as PyData.

What Do They Do? First and foremost, the PyData community has been building a variety of tools (typically released as Python packages). These tools seek to solve data problems (e.g., Web crawling, data acquisition, extraction), implement cross-cutting techniques (e.g., learning, visualization), and help users manage their work (e.g., Jupyter notebook). As of May 2018, there are 138,000+ packages available on $pypi.org$ (compared to "just" 86,000 packages in August 2016). Popular packages include NumPy (49M downloads), pandas (27.7M downloads), matplotlib (13.8M downloads), scikit-learn (20.9M downloads), jupyter (4.9M downloads), etc.

The community has also developed extensive software infrastructure to build tools, and ways to manage/package/distribute tools. Examples include nose, setuptools, $pypi.org$, anaconda, conda-forge, etc. They have also been extensively educating developers and users, using books, tutorials, conferences, etc. Recent conferences include PyData (with many conferences per year, see $pydata.org$), JupyterCon, AnacondaCon, and more. Universities also often hold many annual Data Carpentry Workshops (see $datacarpentry.org$) to train students and scientists in working with PyData.

Finally, the PyData community has fostered many players (companies, non-profits, groups at universities) to work on the above issues. Examples include Anaconda Inc (formerly Continuum Analytics, which releases the popular anaconda distribution of selected PyData packages), NumFocus (a non-profit organization that supports many PyData projects), $datacarpentry.org$ (building communities teaching universal data literacy), $software-carpentry.org$ (teaching basic lab skills for research computing), and more.

Why Are They Successful? Our experience suggests four main reasons. The first obvious reason is that PyData tools are free and open-source, making it cheap and easy for a wide variety of users to use and customize. Many domain scientists in particular prefer open-source tools, because they are free, easy to install and use, and can better ensure transparency and reproducibility (than "blackbox" commercial software). The second reason, also somewhat obvious, is the extensive community effort to assist developers and users, as detailed earlier. The third, less obvious, reason is that PyData tools are *practical*, i.e., they often are developed to address *creators' pain points*. Other users doing the same task often have the same pain points and thus find these tools useful.

Finally, the most important reason, in our opinion, is *the conscious and extensive effort to develop an ecosystem of interoperable tools and the ease of interoperability of these tools*. As discussed earlier, solving DI problems often requires many capabilities (e.g., exploration, visualization, etc.). No single tool today can offer all such capabilities, so DI (and DS) is often done by using a set of tools, each offering some capabilities. For this to work, *tool interoperability is critical*, and PyData appears to do this far better than any other DI platforms, in the following ways. (a) The interactive Python environment makes it easy to interoperate: one just has to import a new tool and the tool can immediately work on existing data structures already in memory. (b) Tool creators understand the importance of interoperability and thus often consciously try to make tools easy to interoperate. (c) Much community effort has also been spent on making popular tools easy to interoperate. For example, for many years Anaconda Inc. has been selecting the most popular PyData packages (536 as of May 2018), curating and making sure that they can interoperate well, then releasing them as the popular anaconda data science platform (with over 4.5M users, as of May 2018).

What Are Their Problems? From DI perspectives we observe several problems. First, even though PyData packages cover all major steps of DI, they are very weak in certain steps. For example, there are many outstanding packages for data acquisition, exploration (e.g., using visualization and statistics), and transformation. But until recently there are few good packages for string matching and similarity join, schema matching, and entity matching, among others. Second, there is very little guidance on how to solve DI problems. For example, there is very little published discussion on the challenges and solutions for missing values, string matching,

entity/schema matching, etc. Third, most current PyData packages do not scale to data larger than memory and to multicore/machine cluster settings (though solutions such as Dask have been proposed). Finally, building data tools that interoperate raises many challenges, e.g., how to manage metadata/missing values/type mismatch across packages. Currently only some of these challenges have been addressed, in an ad-hoc fashion. Clearly, solving all of these problems can significantly benefit from the deep expertise of the DI research community.

5 The Proposed System Building Agenda

Based on the extensive discussion in the previous section, we propose that our community consider extending the PyData "system" to solve DI problems for its users. First, this "system" already exists, with millions of users. So it is important that we consider helping them. (A similar argument was made for XML, MapReduce, etc: regardless of what one thinks about these, many users are using them and we ought to help these users if we can.) Second, we believe that the system building template of PyData is worth exploring. As mentioned earlier, no single uber-system can provide all DI capabilities. An ecosystem of interoperable DI tools situated in an interactive environment (e.g., Python) seems highly promising.

Third, extending PyData brings numerous practical benefits and helps maximize our impacts. (a) By building on PyData, we can instantly leverage many capabilities, thus avoid building weak research prototypes. (b) Many domain scientists use this "system". So by extending it, we can better convince them to use our tools. (c) We can teach PyData and our tools seamlessly in classes, thereby educating students in practical DS tools (that they should know for later jobs) yet obtaining an evaluation of our tools. (d) For our students, developing Python packages are much easier compared to developing complex stand-alone systems (such as RDBMSs). So academic researchers stand a better chance of developing "systems components" that are used in practice. (e) Tools developed by different DI groups have a better chance of being able to work together. (f) PyData is weak in many DI capabilities. So this is a real opportunity for our community to contribute. (g) Finally, if the whole community focuses on a single system, we have a better chance of measuring progress.

Specifically, we propose the following concrete agenda:

- Build systems, each of which helps power users solve an end-to-end scenario involving a single core DI problem, as software packages in PyData.
- Foster PyDI, an ecosystem of such DI software packages as a part of PyData, focusing on how to incentivize the community to grow PyDI, how to make these packages seamlessly interoperate, and how to combine them to solve larger DI problems.
- Extend PyDI to the cloud, collaborative (including crowdsourcing), and lay user settings.

We now motivate and discuss these directions.

5.1 Build Systems for Core DI Problems

To build these systems, we propose the following steps (see Figure 2 for the proposed architecture).

1. Identify a Concrete End-to-End DI Scenario: We propose to start by identifying a concrete scenario that involves *a single core DI problem*. (Later, as we know much better how to solve core DI problems, we can leverage that knowledge to build "composite systems" to jointly solve multiple core DI problems.) This concrete scenario must be *end-to-end (e2e)*, i.e., going from raw data to a desired outcome for the end user (e.g., EM with at least 95% precision and 90% recall). Our goal is to solve this end-to-end scenario for *power users*: those who may not be DI experts but can code, e.g., data scientists. (Later we can leverage these solutions to develop solutions for *lay users*, i.e., those that cannot code, in a way analogous to building on assembly languages to develop higher-level CS languages.)

2. Develop a Detailed How-To Guide: Next, we should develop a detailed how-to guide to the user on how to execute the above e2e scenario, step by step. First, the guide should tell the user whether he or she should execute the scenario in stages. As discussed earlier, there are typically two stages to consider: development

and production. Depending on the scenario, these stages may be executed separately or interleaved somehow. For example, if the user wants to match two tables of 1M tuples each, then the best course of action is to do a development stage first with data samples to find an accurate EM workflow, then execute this workflow on the two original tables in a production stage. On the other hand, if the two tables have just 300 tuples each, then the user can execute the two stages interleavingly "in one shot".

Then for each stage, the guide should tell the user as clearly as possible how it should be executed, step by step: which step should be first, second, etc. For each step, the guide in turn should provide as detailed instruction as possible on how to execute it. In short, the guide is a detailed *algorithms* for the human user. A "rule of thumb" is that if the user knows how to code, he or she should be able to use the guide to execute the e2e scenario, *even without utilizing any tool* (of course, this can take a long time, but the key is that the user should be able to do it). In practice, the guide can utilize any appropriate (semi-)automatic existing tools.

3. Identify Pain Points in the How-To Guide: Next, we should examine the guide carefully to identify real pain points, i.e., steps that are laborious or time consuming for the user and can be fully or partly automated. These include well-known pain points (e.g., performing blocking in EM) or steps that are thought trivial but turn out to be major problems in practice (e.g., browsing a large table, managing missing values, labeling, see Example 3) .

4. Develop Solutions and Tools for the Pain Points: Finally, we should develop algorithmic (semi-)automated solutions for the pain points, then implement them as tools. Each tool is a set of Python packages, often using other packages in PyData. For example, tools for the pain points of the development stage can use packages in the Python Data Analysis Stack (e.g., pandas, scikit-learn, numpy, etc.), while tools for the production stage, where scaling is a major focus, can use packages in the Python Big Data stack to perform MapReduce (e.g., Pydoop, mrjob), Spark (e.g., PySpark), and parallel/distributed computing in general (e.g., pp, dispy).

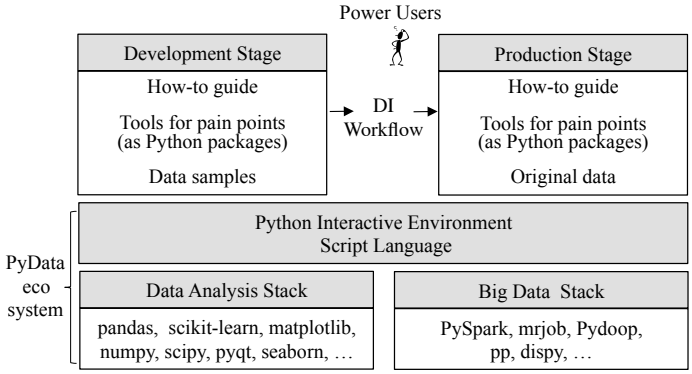


Figure 2: Proposed architecture for DI systems.

Example 4: *We have built* Magellan*, a system that helps users solve the EM problem scenario described in Example 1 [17]. We provide detailed how-to guides, then develop tools for the pain points. For the development stage, these tools (a) take a sample from two tables A and B (ensuring a reasonable number of matches in the sample), (b) debug the blockers, (c) debug the labeling process, (d) select the best matcher, and (e) debug the matchers, among others [24, 27, 28]. There are 104 Python commands that users can use. By leveraging 11 packages in the PyData ecosystem, we were able to develop these tools with relatively little effort. For the production stage, we have developed a preliminary how-to guide, which tells the user how to scale. But more work is necessary to refine the guide.*

We have also been building DI systems to normalize attribute values, count the number of missing values in a column, taking into account synonyms for missing values, and help users clean a table, using a detailed how-to guide which tells him or her which attribute to clean in which order, and how.

5.2 Foster a DI Ecosystem as a Part of PyData

In the second part of our system building agenda, we propose to foster PyDI, an ecosystem of interoperable DI software packages as a part of PyData. To do so, there are several obvious actions we should take. (1) PyData has been very successful. We should study it and similar ecosystems (e.g., R, Bioconductor), then apply the

lessons learned to grow PyDI. (2) We should solve a wide variety of end-to-end DI problem scenarios and build many more tools as Python packages, as discussed earlier. (3) We should work extensively on helping PyDI researchers, developers, educators and users, in terms of infrastructure, books, conferences, tutorials, community resources, etc. To recruit more developers and users, we can apply the lessons learned from PyData/R, "piggyback" on their efforts (e.g., promoting our PyDI tools in Data Carpentry Workshops), and recruit data science students and domain scientists as our users.

Must Ensure That Tools Can Interoperate: Critical to our success of fostering PyDI, however, is the issue of tool interoperability. As described earlier, the proposed DI systems (i.e., tools) will be in the PyData ecosystem and expected to "play well" with other packages. We say that these systems are "open-world", in contrast to current stand-alone "closed-world" DI systems. It is critical that we design these "open-world" DI systems from scratch for interoperability, so that they can easily exploit the full power of PyData and can be seamlessly combined to solve multiple DI problems.

In particular, these systems should expect other systems (in the ecosystem) to be able to manipulate their own data, they may also be called upon by other systems to manipulate those systems' data, and they should be designed in a way that facilitates such interaction. This raises many interesting technical challenges. For example, what kinds of data structures should a tool T use to facilitate interoperability? How to manage metadata if any external tool can modify the data of T, potentially invalidating T's metadata without T knowing about it? How to manage missing values, data type mismatches, version incompatibilities, etc. across the packages? See [17] for a discussion of some of these issues. We should identify and develop solutions for these challenges.

Finally, the PyData experience suggests that it might not be sufficient to just rely on individual creators to make the tools interoperate. Community players may have considerably more resources for cleaning/combining packages and making them work together well. So it is important that we foster such players (e.g., startups, non-profits, research labs, data science institutes, etc.).

5.3 Build Cloud/Collaborative/Lay User Versions of DI Systems

So far we have proposed to develop DI systems for a single power user, in his/her local environment. There are however increasingly many more DI settings, which can be characterized by *people* (e.g., power user, lay user, a team of users, etc.) and *technologies* (e.g., cloud, crowdsourcing, etc.). For instance, a team of scientists wants to solve a DI problem collaboratively, or a lay user wants to do cloud-based DI because he/she does not know how to run a local cluster. To maximize the impacts of our DI systems, we should also consider these settings. In particular, we briefly discuss three important settings below, and show that these settings can build on DI systems proposed so far, but raise many additional R&D challenges.

Cloud-Based DI Systems: Many recent efforts to push PyData "into the cloud" have developed *infrastructure as a service (IaaS)*. Such a service allows a scientist to quickly create a *cloud environment for data science*, by renting machines then installing a wide variety of data science software, including PyData packages (many of which may already been pre-installed). Scientists can then collaborate in this cloud environment. There have been far fewer efforts, however, to develop *software as a service (SaaS)* and *platform as a service (PaaS)* to solve DI problems. We should develop such services. If we can develop cloud services to perform DI tasks (e.g., profiling, cleaning, labeling, matching, etc.), that would bring major benefits. For example, scientists can easily collaborate (e.g., in labeling a dataset), share data (e.g., among team members in multiple locations), remotely monitor task progress, elastically rent more machines/memory to scale, and get access to resources not available locally (e.g., GPU), among others. Further, if we can develop cloud *platform* services for DI, they can save developers a lot of work when building new cloud software services for DI.

Systems for Lay Users: Another promising direction is to customize DI systems discussed so far for lay users (who do not know how to code), by adding GUIs/wizards/scripts as a layer on top. A lay-user action on a GUI, for example, is translated into commands in the underlying system (in a way analogous to translating a Java statement into assembly code). A key challenge is to build this top layer in a way that is easy for lay users to use

yet maximizes the range of tasks that they can perform.

Collaborative Systems: Similarly, we can try to extend the DI systems discussed so far to collaborative settings (including crowdsourcing). This raises many interesting challenges, e.g., how can users (who are often in different locations) collaboratively label a sample and converge to a match definition along the way? How can they collaboratively debug, or clean the data? How can power users, lay users, and possibly also crowd workers work together?

6 Integrating Research, Education, and Outreach

We now discuss how extending PyData allows us to pursue an *integrated* agenda of research, system development, education, and outreach, which in turn can position our community to be a key player in data science.

First, pursuing the above system building agenda raises numerous research challenges. Examples include how to develop the individual tools? How to make them interoperate? How to scale and handle data bigger than memory? How to manage DI workflows (e.g., scaling, incremental execution, what-if execution, provenance)? How to build cloud/collaborative/lay user systems, and many more. Research challenges in building current DI systems would still arise here. But we also have additional challenges that are unique to the PyData setting (e.g., how to make packages interoperate and how to scale across packages?).

Second, we can teach students in DS or DI classes DI principles, solutions, and tools drawn from PyData and PyDI, in a seamless fashion. This helps them gain deep knowledge about DI, but also trains them in tools that they are likely to use in later jobs. Further, many workshops that train domain scientists to do DS already teach PyData, so it is natural to incorporate teaching PyDI.

Third, to make PyDI successful, we need to work with real users and real data. A promising solution for academic researchers is to talk with *domain scientists* at the same university. Twenty years ago this might have been difficult, because most of the data was still at companies. But the situation has dramatically changed. At virtually any university now, often within a walking distance from the CS department, there are many domain science groups that are awash in data, with many pressing DI problems to solve. Since many such groups have been using PyData, we believe they would be willing to try PyDI.

Finally, given that DI is a major part of the DS pipeline, if we can do DI well, via system building, education/training, and outreach to domain sciences, our community should be in a good position to be a key player in data science, by "owning" the DI part (i.e., the data quality part) of this new field.

7 Ongoing Work at Wisconsin

We now describe ongoing work at Wisconsin based on the above agenda, and the main observations so far.

Build Systems for Core DI Problems: We found that this agenda could be used to effectively build a variety of DI systems. Back in 2014 we spent a year building an initial version of Magellan, a Java-based stand-alone EM system, following common practice: the system translates (GUI/command-based) user actions into a workflow of pre-defined operators, then optimizes and executes the workflow. We had serious difficulties trying to extend the system in a clean way to cope with the messiness of real-world EM tasks, where iterations/subjective decisions are the norm, and where exploratory actions (e.g., visualizing, debugging) are very common but it is not clear where to place them in the translate-optimize-execute-workflow paradigm.

Once we switched to the current agenda, these difficulties cleared up. We were able to proceed quickly, and to flexibly extend Magellan in many directions [9]. Using PyData allowed us to quickly add a rich set of capabilities to the system. As far as we can tell, Magellan is the most comprehensive open-source EM system today, in terms of the number of features it supports. Besides EM, we found that the same methodology could also be used to effectively build systems for attribute value normalization and string similarity joins. As discussed earlier, we are also currently working on managing missing values and cleaning tables.

Fostering PyDI: We have been working on position and experience papers [9, 18], instructions on how to

develop PyData packages, datasets, and challenge problems. We have also been partnering with Megagon Labs to encourage a community around BigGorilla, a repository of data preparation and integration tools [4]. The goal of BigGorilla is to foster an ecosystem of such tools, as a part of PyData, for research, education, and practical purposes.

Build Cloud/Collaborative/Lay User Versions of DI Systems: We have developed CloudMatcher, a cloud EM service that is well suited for lay users and crowd workers (e.g., when relying solely on crowdsourcing, it can match tables of 1.8M-2.5M tuples at the cost of only $57-65 on Mechanical Turk) [14]. This is based on our earlier Corleone/Falcon work [13, 7]. We are also developing cloud services for browsing large tables and for collaborative labeling.

Integrating Research, Education, and Outreach: Research-wise we have developed a solution for debugging the blocking step of EM [24] and matching rules [28], using deep learning to match textual and dirty data [27], scaling up EM workflows [7], and developing a novel architecture for cloud EM services [14], among others. We have designed and taught two "Introduction to Data Science" courses, at undergraduate and graduate levels, in multiple semesters. In both courses, students are asked to perform multiple end-to-end DI scenarios, using real-world data and PyData tools, including our PyDI tools. For example, the Magellan system has been used in 5 classes so far, by 400+ students. An extensive evaluation of Magellan by 44 students is reported in [17]. For outreach, Magellan has been successfully used in five domain science projects at UW-Madison (in economics, biomedicine, environmental science [18, 21, 30, 1]), and at several companies (e.g., Johnson Control, Marshfield Clinic, Recruit Holdings [4], WalmartLabs). For example, at WalmartLabs it improved the recall of a deployed EM solution by 34%, while reducing precision slightly by 0.65%. The cloud EM service CloudMatcher [14] is being deployed in Summer 2018 at American Family Insurance for their data science teams to use.

Data Science: Building on the above efforts, in the past few years we have worked to become a key player in data science at UW-Madison, focusing on *data quality* challenges. Specifically, we have been working on developing an UW infrastructure (including on-premise and cloud tools) to help UW scientists solve DI challenges, designing DS courses and programs (which incorporate DI), training UW scientists in DI, providing consulting services in DI, and working with domain sciences (as described above).

Overall, we have found that the proposed agenda is highly promising, synergistic, and practical. As a small example to illustrate the synergy: it is often the case that we conducted research to develop a how-to guide for solving a DI scenario, used it in a class project to see how well it worked for students and refined it based on feedback, then used the improved how-to guide in our work with domain scientists, who can provide further feedback, and so on.

8 Conclusions

In this paper we have discussed the limitations of current DI systems, then proposed an integrated agenda of research, system building, education, and outreach, which in turn can position our community to become a key player in data science. Finally, we have described ongoing work at Wisconsin, which shows the promise of this agenda. More details about this initial work can be found at sites.google.com/site/anhaidgroup.

References

[1] M. Bernstein et al. MetaSRA: normalized human sample-specific metadata for the sequence read archive. *Bioinformatics*, 33(18):2914–2923, 2017.

[2] P. A. Bernstein and L. M. Haas. Information integration in the enterprise. *Commun. ACM*, 51(9):72–79, 2008.

[3] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[4] C. Chen et al. Biggorilla: An open-source ecosystem for data preparation and integration. In *IEEE Data Eng. Bulleting. Special Issue on Data Integration*, 2018.

[5] P. Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012.

[6] X. Chu et al. Distributed data deduplication. In *PVLDB*, 2016.

[7] S. Das et al. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, 2017.

[8] D. Deng et al. The data civilizer system. In *CIDR*, 2017.

[9] A. Doan et al. Human-in-the-loop challenges for entity matching: A midterm report. In *HILDA*, 2017.

[10] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[11] X. L. Dong and D. Srivastava. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool, 2015.

[12] M. J. Franklin, A. Y. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[13] C. Gokhale et al. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.

[14] Y. Govind et al. Cloudmatcher: A cloud/crowd service for entity matching. In *BIGDAS*, 2017.

[15] J. Heer et al. Predictive interaction for data transformation. In *CIDR*, 2015.

[16] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4):281–393, 2015.

[17] P. Konda et al. Magellan: Toward building entity matching management systems. In *VLDB*, 2016.

[18] P. Konda et al. Performing entity matching end to end: A case study. 2016. Technical Report, `http://www.cs.wisc.edu/~anhai/papers/umetrics-tr.pdf`.

[19] S. Krishnan et al. PALM: machine learning explanations for iterative debugging. In *HILDA*, 2017.

[20] S. Kruse et al. Efficient discovery of approximate dependencies. In *PVLDB*, 2018.

[21] E. LaRose et al. Entity matching using Magellan: Mapping drug reference tables. In *AIMA Joint Summit*, 2017.

[22] G. Li. Human-in-the-loop data integration. In *PVLDB*, 2017.

[23] G. Li et al. Crowdsourced data management: A survey. In *ICDE*, 2017.

[24] H. Li et al. Matchcatcher: A debugger for blocking in entity matching. In *EDBT*, 2018.

[25] C. Lockard et al. CERES: distantly supervised relation extraction from the semi-structured web. In *CoRR*, 2018.

[26] A. Marcus et al. Crowdsourced data management: Industry and academic perspectives. In *Foundations and Trends in Databases*, 2015.

[27] S. Mudgal et al. Deep learning for entity matching: A design space exploration. In *SIGMOD*, 2018.

[28] F. Panahi et al. Towards interactive debugging of rule-based entity matching. In *EDBT*, 2017.

[29] O. Papaemmanouil et al. Interactive data exploration via machine learning models. In *IEEE Data Engineering Bulletin*, 2016.

[30] P. Pessig. Entity matching using Magellan - Matching drug reference tables. In CPCP Retreat 2017. `http://cpcp.wisc.edu/resources/cpcp-2017-retreat-entity-matching`.

[31] R. Pienta et al. Visual graph query construction and refinement. In *SIGMOD*, 2017.

[32] F. Psallidas et al. Smoke: Fine-grained lineage at interactive speed. In *PVLDB*, 2018.

[33] T. Rekatsinas et al. Holoclean: Holistic data repairs with probabilistic inference. In *PVLDB*, 2017.

[34] S. W. Sadiq et al. Data quality: The role of empiricism. In *SIGMOD Record*, 2017.

[35] R. Singh et al. Synthesizing entity matching rules by examples. In *PVLDB*, 2017.

[36] M. Stonebraker et al. Data curation at scale: The Data Tamer system. In *CIDR*, 2013.

[37] X. Wang et al. Koko: A system for scalable semantic querying of text. In *VLDB*, 2018.

[38] D. Xin et al. Accelerating human-in-the-loop machine learning: Challenges and opportunities. In *CoRR*, 2018.

[39] M. Yu et al. String similarity search and join: a survey. In *Frontiers Comput. Sci.*, 2016.

Explaining Data Integration

Xiaolan Wang Laura Haas Alexandra Meliou
University of Massachusetts, Amherst
{xlwang, lmhaas, ameli}@cs.umass.edu

Abstract

Explanations are an integral part of human behavior: people provide explanations to justify choices and actions, and seek explanations to understand the world around them. The need for explanations extends to technology, as semi-automated and fully-automated systems support crucial activities and increasingly important societal functions. The interpretability of these systems and the ability to explain their decision processes are crucial in developing trust in the systems' function. Further, explanations provide opportunities for systems to interact with human users and obtain feedback, improving their operation. Finally, explanations allow domain experts and system developers to debug erroneous system decisions, diagnose unexpected outcomes, and improve system function. In this paper, we study and review existing data integration systems with respect to their ability to derive explanations. We present a new classification of data integration systems by their explainability *and discuss the characteristics of systems within these classes. We review the types of explanations derived by the various data integration systems within each explainability class. Finally, we present a vision of the desired properties of future data integration systems with respect to explanations and discuss the challenges in pursuing this goal.*

1 Introduction

Human perception of and reliance on explanations shape all aspects of human activity and our interactions with the world: people rely on explanations to make decisions, justify actions, predict events, and understand the world around them [35, 36]. At the same time, advances in technology and the big data revolution have fundamentally impacted human activity and interactions: data and algorithms play a major role in product recommendations, news personalization, social media interactions, autonomous vehicle decisions, and even medical diagnosis and treatment. The computer systems supporting these tasks are becoming increasingly complex, and their function and decision processes are often poorly understood, even by domain experts. This obscurity is detrimental to user trust in the system operation, and can potentially hinder adoption.

Explanations can significantly ease the interaction between humans and systems: they help humans understand, justify, and consequently trust system function, as well as provide humans with support to debug, diagnose, and improve the systems. This has led to a strong push in several research domains to develop support for explanations in computing systems, such as interpretable machine-learning [23], and DARPA's explainable AI initiative [30]. In the data management community we have also seen a fruitful line of research focusing on supporting explanations in relational [60, 51, 43] and non-relational systems [15].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In this paper, we focus on explanations in data integration systems. Data integration is a crucial component in data analytics and data management applications, involving the acquisition, storage, and management of data gathered from heterogeneous data sources, through a uniform query interface [38]. Unlike fully-automated systems, data integration systems often interact with humans extensively. Many existing data integration systems are semi-automated and require several iterations with human input. Even automated data integration tasks may still rely on humans to remove the uncertainty from the results (e.g., in schema mapping). Support for explanations in data integration systems facilitates the interactions between these systems and human users, improving not only the user experience, but also the overall quality of these systems' function.

In this paper, we offer a review of existing data integration research and systems with respect to their ability to provide explanations. We present a new classification of the existing work into three *explainability* categories, and discuss the characteristics of each class (Section 2). We review existing solutions for four core data integration tasks with respect to our classification (Section 3). We then conduct detailed comparison and analysis over existing explanations in data integration systems (Section 4). Finally, we present a vision of desirable explanation properties for future data integration systems and discuss the challenges in pursing this goal (Section 5).

2 Classifying data integration systems' explainability

In this section, we present a new classification of data integration systems with respect to their explainability and the type of explanations they produce. In general, explanations can be categorized into two major classes: (1) *causal* explanations typically focus on *how and why* an outcome was derived; (2) *non-causal* explanations typically focus on *what* the outcome is. For example, for the schema matching pair `(class, course)`, the explanation *"more than* 90% *of the values in attribute* `class` *and attribute* `course` *match"* is causal: it explains why these attributes were matched. In contrast, the explanation, *"attribute* `class` *in schema A and attribute* `course` *in schema B match with each other"* is non-causal: it explains what the schema matching output represents, but it does not reveal how or why it was derived. Some data integration systems focus on causal explanations as an *explicit* objective. However, causal explanations may also be *implicitly* derived from systems that do not target them directly. Systems that require human involvement often focus on non-causal explanations to make their results understandable to and allow for feedback from human users. Finally, a large number of data integration systems do not derive explanations of any kind (unexplainable systems). Figure 1 provides an overview of our classification of data integration systems. We proceed to describe our classification and discuss the characteristics of systems within each class.

Explaining systems

Our first class includes methods and systems that focus on providing causal explanations as an explicit objective. We call this class *explaining systems*. Such systems build connections between evidence and results in an attempt to explain *how* and *why* particular results are derived. The purpose of some of these systems is only to derive explanations of a data integration process, and they treat data integration results as part of their input. Other systems aim to derive results, as well as their explanations, at the same time. Explaining systems often use conciseness as an objective and performance metric in generating and summarizing explanations. As a result, the causal explanations they produce tend to be simple and understandable, and thus easily accessible to human users. Explaining systems produce explanations that allow users to understand and validate results, and engineers to debug, diagnose, and fix problems in the systems.

Explainable systems

Outside of the explaining systems class, causal explanations are not an explicit goal. The majority of data integration systems focus on other goals and metrics, typically, higher accuracy, scalability, and efficiency.

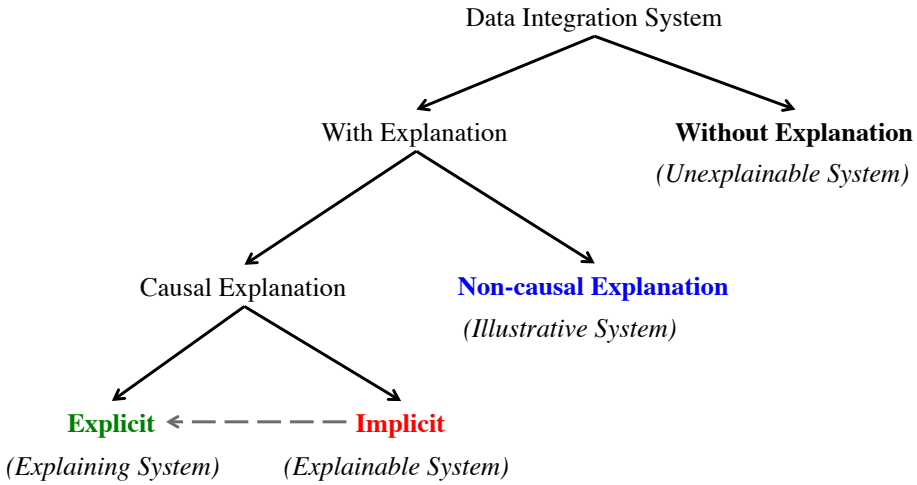


Figure 1: Explainability classification for data integration systems

However, causal explanations may still be present implicitly in approaches that do not target them directly, derived as a by-product of the system processes. These systems comprise the *explainable systems* class. In this class, since explanations are not an explicit objective, the explanations produced are typically suboptimal with respect to common explanation quality metrics, such as conciseness. Frequently, these explanations are complex, long, and hard or even impossible for humans to interpret and digest. However, there is potential to improve these explanations in a post-processing step, which summarizes and abstracts them. Such a transformation would convert explainable systems to explaining systems.

Illustrative systems

Our third class, *illustrative systems*, includes systems that provide non-causal explanations for their results or processing steps. Non-causal explanations are common in data integration systems that prioritize human involvement as a way to improve system function and result quality. In these systems, non-causal explanations, such as simplification, examples, and visual demonstrations, are often used to ease the processes of tapping into human knowledge as an information source and employing human intelligence to solve data integration tasks. For example, non-causal explanations make it possible for a human user to provide feedback for a schema mapping application by simplifying the possible mappings and demonstrating each mapping through examples. Note that human-in-the-loop systems are not necessarily illustrative systems: some human-in-the-loop systems may provide causal explanations and some may not provide any kind of explanation.

Unexplainable systems

The remaining data integration systems are unexplainable systems. Unexplainable systems often use highly complex procedures or purpose-built code to generate results and they do not interact with humans during the integration process. Unexplainable systems may derive high quality results extremely efficiently, but they can be hard to debug and analyze. Note that although many data integration systems, especially those with probabilistic-based approaches, are currently unexplainable, advances in explainable AI [30] could impact their explainability classification in the future and may potentially transform unexplainable systems into explainable ones. We will not discuss this class of systems further in this paper.

		Explaining systems	Explainable systems	Illustrative systems
Schema Level	**schema matching**	iMap [17], S-Match [28, 53] mSeer [9]	CUPID [42], Peukert et al. [48], YAM++ [45], AutoMatch [5]	AgreementMaker [13, 14] COMA++ [3], CCQ [63]
	schema mapping	Spider [1, 12], TRAMP [27]	RiMOM [56], CMD [37]	Clio [44, 62], Muse [2], Tupelo [25], Mweaver [50], Prompt [47]
Data Level	**record linkage entity resolution deduplication**	None	Davis et al. [16], Whang et al. [57], HIL [32], Li et al. [40], ActiveAtlas [54, 55]	Alias [52], GCER [58], DIMA [41], C3D+P [11], D-Dupe [8, 34]
	data fusion truth discovery	Dong & Srivastava [22], Popat et al. [49], PGM [46]	VOTE [18], Wu & Marian [59]	CrowdFusion [10]

Table 1: Explainability classification of core techniques in data integration.

3 Data integration tasks through the lens of explainability

The field of data integration has expanded in many directions [31, 21, 29], including tasks such as schema matching, schema mapping, record linkage, data fusion, and many more. In this section, we study and review existing approaches for four core data integration tasks, two for schema level integration and two for data level integration, and summarize them based on the classification of Table 1.

3.1 Schema matching

Schema matching, a fundamental requirement for schema mapping, focuses on finding the correspondence among schema elements in two semantically correlated schemata. Existing schema matching solutions leverage a wide variety of techniques, from heuristics, to rules, to learning-based approaches. These techniques give them different abilities to provide explanations. Here we discuss some representative systems in each of the three explainability classes.

Explaining systems. Explaining systems for schema matching have various goals, such as result justification or system debugging. **iMap** [17] is a semi-automated schema matching system that leverages users to finalize matching decisions. To assist users in making matching decisions, iMap provides insight into the derived matches by presenting *causal explanations* that are associated with each match. iMap does this by constructing a dependency graph that abstracts the process and the major decisions in deriving a match; the explanations reflect causality through the way they are generated. iMap further visualizes the dependency graph explanations such that they can be easily interpreted by users. **S-Match** [28, 53] uses explanations to gain trust from the users: it has a multi-level explanation module that generates high-level, concise explanations, as well as more verbose explanations. A high-level explanation is typically a short piece of natural language without any technical detail about the matching process; verbose explanations exploit both background knowledge and the logical reasoning of the matching process. **mSeer** [9] is an analytic tool that uses explanations to debug schema matching systems. mSeer does not generate matches itself. Instead, it takes the output of a matching system and produces a concise matching report, in which it displays all the derived matches, and the aggregated positive (e.g., matchability statistics) and negative (e.g., common matching mistakes) evidence for each match. iMap, S-Match, and mSeer belong to the class of explaining systems as they all treat deriving causal explanations as an explicit goal:

S-Match and iMap explain how a match is derived, and mSeer explores why a match is true (or false).

Explainable systems. Rule-based techniques, such as **CUPID** [42] and **Peukert et al.** [48], are known to be explainable and interpretable as rules are often self-explaining. For example, a rule "two schema elements match with each other if they are semantically similar" directly explains why two attributes are considered as a match. Machine-learning techniques have also been widely applied to schema matching problems. Some of these machine-learning techniques are likewise known to be explainable and interpretable, such as decision tree [26] and naïve Bayes [39]. For example, **YAM++** [45] leverages decision trees to combine different terminological similarity measures in order to justify the candidate matches. In each decision tree, non-leaf nodes represent the similarity measures and the leaf nodes indicate whether the match is valid or not. Thus, the matching decision is explainable by tracing the path through the decision tree. **AutoMatch** [5] uses naïve Bayes analysis and data samples of the attributes to determine the matching relationship. Their results are explainable since the conditional probability for each matching decision is calculated independently and the probability value directly reflects its contribution to the final matching decision. The key factor that distinguishes the above systems from explaining systems is that they do not produce explanations as part of their output. In addition, their explanations are often hard to understand as they do not restrict the number of rules ([48]), the size of the decision tree (YAM++), or the number of data samples (AutoMatch).

Illustrative systems. Due to the complexity and intricacy of the schema matching process, researchers have realized the limitations (in terms of precision and recall performance) of fully-automated schema matching techniques [24]. Therefore, many schema matching systems require human intervention and user validation [17, 63, 14, 47, 3, 33] to further refine their results. Most of these human-in-the-loop systems will iteratively ask users to accept or reject a selected set of candidates and then update their matching results accordingly. Meanwhile, they also provide a variety of non-causal explanations to facilitate these interactive processes. Some systems visualize the matches through user interfaces in order to help users understand the matches. Such systems include **AgreementMaker** [13, 14], and **COMA++** [3]. Another thrust of such systems focuses on leveraging the intelligence of the crowd, instead of domain experts, to answer the questions (e.g., **CCQ** [63]). Since the crowd often has little or even no background knowledge, these crowd-based systems typically simplify their questions to make the requested matching feedback understandable and accessible. Instead of explaining how the results are produced, illustrative systems in schema matching focus on explaining and demonstrating the results themselves through visualization (AgreementMaker, COMA++) or question simplification (CCQ).

3.2 Schema mapping

Schema mapping, a core operation for all data integration systems, describes how a source database schema relates to a target database schema [6, 4]. The mappings, typically generated from the matches between schema elements, form the basis for transforming the data from the source into the target. Schema matching and mapping are two strongly correlated components in data integration, and many schema mapping systems are also equipped with a schema matching module.

Explaining systems. Explaining systems in schema mapping primarily focus on system debugging. **Spider** [1, 12] is a debugging tool for exploring and understanding schema mappings. It does not identify mapping errors, but rather generates *routes* that describe the causal relationship between source and target data with the schema mapping. Thus it provides a platform that allows users to find the errors more easily. On the other hand, **TRAMP** [27] assists users in identifying mapping errors; it uses mapping provenance to explore common mapping errors and report the analysis, together with the evidence, to the users. Both systems consider deriving concise explanations as one of their objectives: Spider ranks the explanations and only returns the top ones; TRAMP always highlights the conclusion of the explanations – the identified errors – in its results.

Explainable systems. As with schema matching techniques, systems that leverage interpretable learning-based approaches are also explainable. For example, **RiMOM** [56] uses naïve Bayes to decide whether there exists

a mapping and, further, the type of the mapping between schema elements. **CMD** [37] poses a new direction for solving schema mapping problems by taking advantage of both rule-based and learning-based approaches. It leverages probabilistic soft logic (PSL), a scalable probabilistic programming language over weighted first-order logic rules, to derive the schema mapping. CMD first expresses the candidate mappings through first-order logic and then infers the weights of the mappings. The results of CMD are explainable since one can learn the contribution of a candidate mapping through learned weights. As with other interpretable learning-based approaches, RiMOM and CMD do not return these explanations, hence they are explainable, but not explaining; moreover, these explanations could be hard to understand due to the complexity and number of the sub-decisions (RiMOM) and rules (CMD).

Illustrative systems. There are many schema mapping techniques that rely on users to reduce the uncertainty of the mapping and to make the final mapping decisions. However, as the mappings are often complex programs, explaining the mappings to the users becomes a critical issue. Some systems, such as **Clio** [44, 62], **Muse** [2], **Tupelo** [25] and **Mweaver** [50], use data examples to explain the mappings; other systems, e.g., **Prompt** [47], rely on graphical interfaces to visualize the mapping between the source schema and the target schema.

3.3 Record linkage

Record linkage (also known as entity resolution and deduplication) is the problem of identifying records that refer to the same logical entity, and it is an important task in integrating data from different data sources. Unfortunately, to the best of our knowledge, *none of the existing record linkage systems explicitly provide causal explanations* of their results. However, many record linkage systems are explainable because of the method they use in determining the matching records.

Explainable systems. Explainable systems for record linkage often rely on approaches that naturally contain explanations. Rule-based record linkage methods typically expose understandable logic. For example, "$same\_name(t_1, t_2) \rightarrow duplication(t_1, t_2)$", intuitively means: "two records refer to the same entity if they have the same name". There are many rule-based record linkage systems, and they often rely on different methods to derive the linkage rules. For example, **ActiveAtlas** [54, 55] uses decision trees to learn the mapping rules over the records; **Whang et al.** [57] and **Li et al.** [40] leverage dynamic rules, learned from an iterative process, for discovering duplicate records; **HIL** [32] applies SQL-like syntax rules to accomplish entity resolution, mapping, and fusion tasks. Meanwhile, some record linkage solutions use explainable probabilistic-based approaches, combined with rules, to resolve the uncertainty in the mappings. Such approaches can be explainable as well. For example, **Davis et al.** [16] is explainable since it combines two explainable components, rules and naïve Bayes, to solve the record linkage problem. Again, the explanations are implicit in the specifications, not produced by the systems, so these are explainable, not explaining systems. Further, these systems usually include many rules, sub-decisions, or explainable components, which increase the overall complexity of their explanations. Therefore, in most cases, their explanations are still hard for humans to understand.

Illustrative systems. Researchers have developed a wide variety of record linkage solutions [52, 8, 58, 11, 41] that employ human intelligence. However, obtaining user input efficiently and effectively remains challenging. Many record linkage systems try to reduce the number of record pairs that need to be justified by humans. **Alias** [52] attempts to limit the manual effort by reducing the number of record pairs according to the information gained from them; **GCER** [58] measures the quality of questions (or record pairs) based on their impact over the system and selects questions that lead to the highest expected accuracy; **DIMA** [41] evaluates the questions according to their impact on the monetary cost and selects those that reduce the cost. We classify these as illustrative systems since they offer, in the form of questions, non-causal examples of possible linkages, and try to reduce the number of such questions to make it easier for the user to understand. **C3D+P** [11] eases the tasks handed to humans along another dimension – it simplifies and summarizes record descriptions by selecting a small subset of critical attributes. **D-Dupe** [8, 34] provides a visual interface that shows the relational

neighborhood for each record in a candidate pair to facilitate users' decisions.

3.4 Data fusion

Data errors and conflicting information are common occurrences across different data sources. Data fusion focuses on resolving conflicts and determining the true data values, leveraging information in heterogeneous data sources [7, 20].

Explaining systems. Explaining systems for data fusion explore explanations from both structured and unstructured data. **Dong & Srivastava** [22] leverage Maximum A Posteriori (MAP) analysis to make data fusion decisions about data stored in a DBMS. MAP analysis is explainable since a probability value for each candidate decision is calculated independently. However, the raw MAP explanations are complex and excessively detailed; Dong & Srivastava [22] abstract them through categorization and aggregation to produce succinct and understandable explanations for the decisions. Unlike the majority of data integration tools, which focus on structured or semi-structured data, **Popat et al.** [49] fuses data in unstructured format. Instead of fusing conflicting data values, this work tries to determine the credibility of natural language claims and explore the evidence, identified from articles on the Web, to support or refute the claim through Conditional Random Field (CRF) analysis. Explanations in this system are easy to understand as they are all in natural language. In a similar vein, **PGM** [46] also targets discovering the truthfulness of natural language claims. PGM further improves the understandability of the explanations by aggregating the evidence and visualizing them through a user interface.

Explainable systems. Many data fusion systems are based on probabilistic analysis, such as Bayesian analysis [18], probabilistic graphical models [19, 64] and SVM [61], and most of these models are hard to explain to humans with little domain knowledge. **VOTE** [18] discovers true values by incorporating both the data source dependence, i.e. the copying relationship between data sources, and data source accuracy. VOTE is explainable since it relies on several explainable steps: First, it uses Bayesian analysis to calculate the probability of dependence and accuracy among data sources; second, it estimates the vote count of values, combined with the dependence probabilities, through a greedy algorithm; finally, it combines the vote count and accuracy of the data source through a weighted sum formula. **Wu & Marian** [59] use a scoring mechanism, based on the score of the source and scores of values within the data source, to decide the actual result of a query. Similar to other explainable systems, VOTE [18] and Wu & Marian [59] do not derive explanations directly and their explanations may also be too complex to interpret in practice.

Illustrative systems. Incorporating human intelligence in solving data fusion problems is challenging as humans may be misled by conflicting information in data sources and the Web. **CrowdFusion** [10] tackles this challenge by allowing mistakes in the crowd answers: it assigns a probability for the correctness of an answer, and assumes that crowd answers follow a Bernoulli distribution. CrowdFusion adjusts the fusion results upon receiving relevant answers from the crowd and optimizes its questions by maximizing the questions' entropy.

4 Properties of explanations

Explanation coverage and understandability. So far, in this paper, we have categorized explanations in data integration as one of two types: causal or non-causal. A causal explanation is essentially evidence, or a summary of evidence that supports or refutes a data integration outcome. A non-causal explanation does not provide any evidence of the data integration process, but helps users understand the results. In this section, we further examine these explanations along two metrics, coverage and understandability. *Coverage* measures the amount of evidence with respect to the results of a data integration process. More precisely, 100% *coverage* means that one can replicate all decisions of the algorithm purely based on the explanations, whereas *lower coverage* means that one can only replicate a smaller portion of such decisions. *Understandability* intuitively measures the difficulty in understanding the explanations.

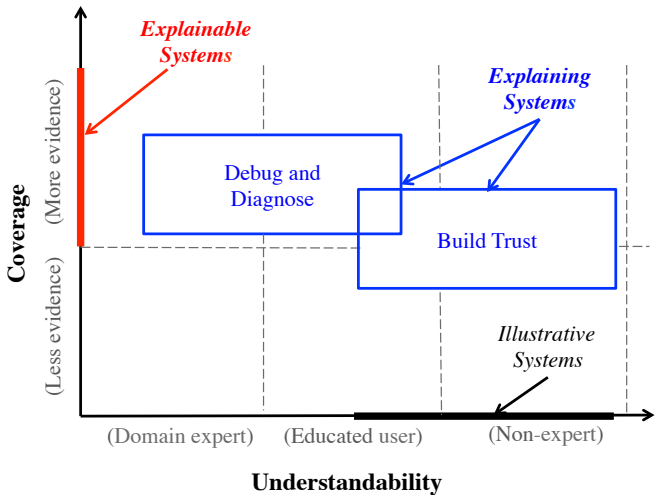


Figure 2: Coverage vs. understandability

Figure 2 presents our three explainability classes with respect to coverage and understandability. Explainable systems have high coverage since every step in their algorithm is interpretable. However, they are typically hard to understand, even for domain experts, since the explanations are usually complex and long. Illustrative systems present the opposite behavior: they do not cover any evidence of the data integration processes—their goal is to explain what the results are, not how they were produced—but are easy to understand by most users. Explaining systems show more diverse behavior depending on their motivation for providing explanations. Systems whose goal is to build trust in the data integration process tend to have lower coverage but higher understandability. In contrast, systems that focus on diagnosis and debugging typically produce explanations with higher coverage but lower understandability, as they target more educated users and domain experts.

We proceed to review these properties in more detail for explaining systems and highlight differences in their objectives and results.

Properties of explaining systems. Explaining systems provide evidence, in the form of explanations, that supports or refutes the results. Such evidence can be found in the input, the algorithm, or the combination of the two, and it is presented in various formats, such as examples, rules, or statistics. Discovering the evidence is fundamental to deriving causal explanations, but many explanation systems explore beyond this point in order to further improve their understandability. Some systems score the importance of the evidence and only report the top indicators; some systems aggregate the evidence and report the aggregated values instead of other details. In Table 4, we enumerate the properties of explanations for explaining systems along five dimensions, including the goal of the explanations, whether the explanations include details of the algorithm, the audience for the explanations, their format, and whether there is any post-processing associated with the discovered evidence.

Although all explaining systems provide causal explanations of their result, their coverage may be different due to differing explanation objectives. Systems that derive explanations for debugging purposes usually provide evidence not only from the input but also from the algorithms in order to reveal enough detail. However, this is not the case for external analytic tools, including mSeer, Spider, and TRAMP. With the goal of justifying the results and debugging the functionality of an existing data integration algorithm, these external analytic tools often conduct independent analysis over a data integration algorithm and reason about the correctness and the causes of the correctness of its results. Meanwhile, systems that use explanations to build trust typically do not reveal many details from the algorithm, and often only highlight evidence discovered from the input. The audience for the explanation correlates with the explanation objectives. Explanations meant for debugging usually target educated users, who have some background knowledge of the problem and algorithm, and may

System	Goal	Algorithm?	Audience	Format	Post-processing?
iMap [17]	Get feedback	Yes	Educated user	Visualization	None
S-Match [28, 53]	Build trust	No	Non-expert	Text	Summarization
	Debug	Yes	Domain expert	Rules	None
mSeer* [9]	Debug	No	Domain expert	Statistics	Summarization
Spider* [1, 12]	Debug	No	Domain expert	Rules	TopK
TRAMP* [27]	Debug	No	Domain expert	Tuples, rules	None
Dong & Srivastava [22]	Build trust	Yes	Non expert	Statistics	Summarization
Popat et al. [49]	Build trust	No	Non expert	Natural Language	Summarization
PGM [46]	Build trust	No	Non expert	Statistics	Summarization

Table 2: Analysis of properties of explaining systems. Highlighted systems (mSeer, Spider, and TRAMP) are external analytic tools; the others are a build-in component of the algorithms.

even be domain experts. In contrast, systems aimed at building trust are designed to be understandable even for non-expert users, with little background knowledge of the problem and algorithm. Even with the same explanation goal, systems may use different explanation formats, or post-process the discovered evidence to further improve the understandability of their explanations. In general, visualizing explanations may help users understand the explanations. In addition, condensing and summarizing the evidence may significantly reduce the volume of the explanations, and therefore improve their understandability.

5 Summary, next steps, and challenges for explaining data integration

In this paper, we presented a new classification of data integration systems by their explainability, and studied the systems and their characteristics within each class. The rich literature on data integration already demonstrates a strong and established focus on explanations. Over the past few years, data integration systems have developed a variety of explanations that serve diverse objectives. For example, human-in-the-loop systems frequently use non-causal explanations to illustrate the meaning of their results, and many explaining systems leverage causal explanations, associated with their results, to gain trust from the users. However, the portion of data integration systems that provide explanations, in particular causal explanations, is still small. For example, causal explanations are not an explicit objective of any existing approach in record linkage. The lack of explainability impedes the usability and evolution of data integration systems: humans cannot understand the results, provide feedback, or identify problems and suggest improvements.

We envision a future generation of data integration systems where *flexible* explanatory capabilities are a primary goal. Our study showed that there is a natural trade-off between coverage and understandability (Figure 2), and it is often driven by the underlying goals of the explanations (e.g., debugging, trust). Explanations that are used for debugging and diagnosis tend to be less understandable, however, they include more details of the data integration process. In contrast, non-causal explanations focus on explaining what the result is, and they typically offer little or no coverage of the process itself. The explanation goals also correlate with the target audience and the explanation type and content. To cater to varied users, uses, and goals, data integration systems need to support ***flexible, diverse, and granular explanation types***. Such systems would seamlessly transition from non-causal explanations facilitating interactions with non-experts, to causal explanations of the integration process facilitating debugging by domain experts. We identify the following challenges in achieving this vision.

Interactive explanations. To allow flexibility in navigating across explanations of different granularities and types, systems need to facilitate user interactions with the explanations. Users should be able to drill down,

generalize, and even ask for explanations of the explanations themselves.

From explainable to explaining. Explainable systems produce rich causal explanation content. However, these explanations are often not exploited, because they are typically complex, long, and hard or even impossible for humans to interpret and digest. External application tools could process, summarize, and abstract these explanations to different granularities, increasing the utility of explainable systems and essentially transforming them into explaining systems.

Evaluating understandability. Understandability is an important factor in formulating explanations, but it is a difficult factor to evaluate and measure. The fundamental challenge is that judgments of understandability are often subjective and could vary among people with different levels of background knowledge. System designers, knowledgeable users, and complete non-experts have different expectations from explanations. An explanation that is understandable and useful for one group may not be for another. To evaluate understandability in practice, researchers will have to rely on user studies with varied target users, ideally producing and generalizing guiding principles in explanation design.

Accuracy vs. explainability. Explaining data integration systems need to balance the tradeoff between accuracy and explainability. Sophisticated machine-learning approaches are widely used for solving data integration tasks. These methods achieve high accuracy, but are typically not explainable. Any solution in this domain will need to consider and balance these often conflicting objectives, to produce systems that are both effective and usable.

References

[1] B. Alexe, L. Chiticariu, and W.C. Tan. Spider: a schema mapping debugger. *VLDB*, 1179–1182, 2006.

[2] B. Alexe, L. Chiticariu, R.J. Miller, and W.C. Tan. Muse: Mapping understanding and design by example. *ICDE*, 10–19, 2008.

[3] D. Aumuller, H.H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. *SIGMOD*, 906–908, 2005.

[4] Z. Bellahsene, A. Bonifati, F. Duchateau, and Y. Velegrakis. On evaluating schema matching and mapping. *In Schema matching and mapping*, 253–291, 2011.

[5] J. Berlin, and A. Motro. Database schema matching using machine learning with feature selection. *In International Conference on Advanced Information Systems Engineering*, 452–466, 2002.

[6] P.A. Bernstein, and L.M. Haas. Information integration in the enterprise. *Communications of the ACM*, 51(9):72–79, 2008.

[7] J. Bleiholder, and F. Naumann. Data fusion. *ACM Computing Surveys (CSUR)*, 41(1):1, 2009.

[8] M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. D-dupe: An interactive tool for entity resolution in social networks. *In Visual Analytics Science And Technology*, 43–50, 2006.

[9] X. Chai, M. Sayyadian, A. Doan, A. Rosenthal, and L. Seligman. Analyzing and revising data integration schemas to improve their matchability. *PVLDB*, 1(1):773–784, 2008.

[10] Y. Chen, L. Chen, and C.J. Zhang. CrowdFusion: A Crowdsourced Approach on Data Fusion Refinement. *ICDE*, 127–130, 2017.

[11] G. Cheng, D. Xu, and Y. Qu. C3d+ p: A summarization method for interactive entity resolution. *Web Semantics: Science, Services and Agents on the World Wide Web*, (35):203–213, 2015.

[12] L. Chiticariu, and W.C. Tan. Debugging schema mappings with routes. *VLDB*, 79–90, 2006.

[13] I. Cruz, F. Antonelli, and C. Stroe. Agreementmaker: efficient matching for large real-world schemas and ontologies. *PVLDB*, 2(2):1586–1589, 2009.

[14] I. Cruz, C. Stroe, and M. Palmonari. Interactive user feedback in ontology matching using signature vectors. *ICDE*, 1321–1324, 2012.

[15] M. Das, S. Amer-Yahia, G. Das, and C. Yu. Mri: Meaningful interpretations of collaborative ratings. *PVLDB*, 4(11), 1063–1074,2011.

[16] J. Davis, I. Dutra, D. Page, and V. Costa. Establishing identity equivalence in multi-relational domains.*In Proc. ICIA*, 2005.

[17] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: discovering complex semantic matches between database schemas.*SIGMOD*, 383–394, 2004.

[18] X.L. Dong, L. Berti-Equille, and D. Srivastava. Integrating conflicting data: the role of source dependence. *PVLDB*, 2(1),550–561, 2009.

[19] X.L. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. *PVLDB*, 2(1):562–573, 2009.

[20] X.L. Dong, and F. Naumann. Data fusion: resolving data conflicts for integration. *PVLDB*, 2(2), 1654–1655, 2009.

[21] X. L. Dong, and D. Srivastava. Big data integration. *ICDE*, 1245–1248, 2013.

[22] X. L. Dong, and D. Srivastava. Compact explanation of data fusion decisions. *WWW*, 379–390, 2013.

[23] F. Doshi-Velez, and B. Kim.Towards a rigorous science of interpretable machine learning. *arXiv*, 2017.

[24] Z. Dragisic, V. Ivanova, P. Lambrix, D. Faria, E. Jiménez-Ruiz, and C. Pesquita, C. User validation in ontology alignment.*In International Semantic Web Conference*,200-217, 2016.

[25] G.H.L. Fletcher, and C.M. Wyss.Data Mapping as Search.*EDBT*,95–111, 2006.

[26] M.A. Friedl, and C.E. Brodley. Decision tree classification of land cover from remotely sensed data. *Remote sensing of environment*, 61(3):399–409, 1997.

[27] B.Glavic, G. Alonso, R.J. Miller, and L.M. Haas. TRAMP: Understanding the behavior of schema mappings through provenance. *PVLDB*, 3(1-2):1314–1325, 2010.

[28] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. *In European semantic web symposium*, 61–75, 2004.

[29] B. Golshan, A. Halevy, G. Mihaila, and W.C. Tan. Data integration: After the teenage years. *SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 101–106, 2017.

[30] D. Gunning. Explainable artificial intelligence (xai).*Defense Advanced Research Projects Agency (DARPA)*, 2017.

[31] L. Haas. Beauty and the beast: The theory and practice of information integration. *ICDT*, 28–43, 2007.

[32] M. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, and R. Wisnesky. HIL: a high-level scripting language for entity integration.*In Proceedings of the 16th international conference on extending database technology*, 549–560, 2013.

[33] E. Jimenez-Ruiz, B. Cuenca Grau, Y. Zhou, and I. Horrocks.Large-scale Interactive Ontology Matching: Algorithms and Implementation.*ECAI*, 444–449, 2012.

[34] H. Kang, L. Getoor, B. Shneiderman, M. Bilgic, and L. Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):999–1014, 2008.

[35] F.C. Keil, and R.A. Wilson. Explaining explanation.*Explanation and cognition*, 1–18, 2000.

[36] F.C. Keil. Explanation and understanding. *Annu. Rev. Psychol.*, 57:227–254, 2006.

[37] A. Kimmig, A. Memory, and L. Getoor. A collective, probabilistic approach to schema mapping. *ICDE*, 921–932, 2017.

[38] M. Lenzerini. Data integration: A theoretical perspective. *SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 233–246, 2002.

[39] B. Letham, C. Rudin, T.H. McCormick, and D. Madigan.Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9(3):1350–1371, 2015.

[40] L. Li, J. Li, and G. Hong. Rule-based method for entity resolution. TKDE, 27(1):250–263, 2015.

[41] G. Li. Human-in-the-loop data integration. *PVLDB*, 10(12):2006–2017, 2017.

[42] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with cupid. *VLDB*, (1):49–59, 2001.

[43] A. Meliou, S. Roy, and D. Suciu. Causality and explanations in databases. *PVLDB*, 7(13), 1715–1716, 2014.

[44] R.J. Miller, L.M. Haas, and M.A. Hernändez. Schema mapping as query discovery. VLDB, (2000):77–88, 2000.

[45] D.H. Ngo, and Z. Bellahsene. YAM++:(not) Yet Another Matcher for Ontology Matching Task. *In BDA: Bases de Données Avancées*, 2012.

[46] A.T. Nguyen, A. Kharosekar, M. Lease, and B.C. Wallace. An Interpretable Joint Graphical Model for Fact-Checking from Crowds. *AAAI*, 2018.

[47] N.F. Noy, and A.M. Mark. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies* 59(6):983–1024, 2003.

[48] E. Peukert, J. Eberius, and E. Rahm. A self-configuring schema matching system. *ICDE*, 306–317, 2012.

[49] K. Popat, S. Mukherjee, J. Strötgen, and G. Weikum. Where the truth lies: Explaining the credibility of emerging claims on the web and social media. *WWW*, 1003–1012, 2017.

[50] L. Qian, M.J. Cafarella, and H.V. Jagadish. Sample-driven schema mapping. *SIGMOD*, 73–84, 2012.

[51] S. Roy, D. Suciu. A formal approach to finding explanations for database queries. *SIGMOD*, 1579–1590, 2014.

[52] S. Sarawagi, and A. Bhamidipaty. Interactive deduplication using active learning. *SIGKDD*, 269–278, 2002.

[53] P. Shvaiko, F. Giunchiglia, P.P. Da Silva, and D.L. McGuinness. Web explanations for semantic heterogeneity discovery. *European Semantic Web Conference*, 303–317, 2005.

[54] S. Tejada, C.A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Special Issue on Data Extraction, Cleaning, and Reconciliation, Information Systems Journal*, 26(8):607–633, 2001.

[55] S. Tejada, C.A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. *SIGKDD*, 350–359, 2002.

[56] J. Tang, J. Li, B. Liang, X. Huang, X., Y. Li, and K. Wang. Using Bayesian decision for ontology mapping. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(4):243–262, 2006.

[57] S.E. Whang, and H. Garcia-Molina. Entity resolution with evolving rules. *PVLDB*, 3(1-2):1326–1337, 2010.

[58] S.E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[59] M. Wu, and A. Marian. A framework for corroborating answers from multiple web sources. *Information Systems*, 36(2):431–449, 2011.

[60] E. Wu, and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8), 553–564, 2013.

[61] X. Yin, and W. Tan. Semi-supervised truth discovery. *WWW*, 217–226, 2011.

[62] L.L. Yan, R.J. Miller, L.M. Haas, and R.Fagin. Data-driven understanding and refinement of schema mappings. *In ACM SIGMOD Record*, 30(2):485–496, 2001.

[63] C.J. Zhang, L. Chen, H.V. Jagadish, and C.C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9):757–768, 2013.

[64] B. Zhao, B.I.P. Rubinstein, J. Gemmell, and J. Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.

Making Open Data Transparent:
Data Discovery on Open Data

Renée J. Miller[1], Fatemeh Nargesian[1]
Erkang Zhu[1], Christina Christodoulakis[1], Ken Q. Pu[2], Periklis Andritsos[1]
[1]University of Toronto, [2]UOIT
{miller,ekzhu,fnargesian,christina}@cs.toronto.edu,
ken.pu@uoit.ca, periklis.andritsos@utoronto.ca

Abstract

Open Data plays a major role in open government initiatives. Governments around the world are adopting Open Data Principles *promising to make their Open Data complete, primary, and timely. These properties make this data tremendously valuable. Open Data poses interesting new challenges for data integration research and we take a look at one of those challenges,* data discovery. *How can we find new data sets within this ever expanding sea of Open Data. How do we make this sea transparent?*

1 Introduction

Governments in many countries have recognized that the provision of open access to their data promotes government transparency and can be beneficial to the public good [3, 34]. Therefore, both federal and local governments have been putting policy frameworks in place in order to enable access to their data. By committing to common Open Data Principles, they are promising to ensure their Open Data is complete, accurate, has primacy, and is released in a timely manner. These properties make this data of great value to data scientists, journalists, and to the public. When Open Data is used effectively, citizens can explore and analyze public resources, which in turn allows them to question public policy, create new knowledge and services, and discover new (hidden) value useful for social, scientific, or business initiatives.

Given the value of Open Data, an important question is how well are we doing in providing adequate tools for finding, using, and integrating Open Data? We take a look at the first task in data integration, *data discovery*. We consider data sets of structured data which may be from a relational DBMS, but more often are extracted from spreadsheets or document collections and published in formats such as CSV or JSON. We will use the term data set and table interchangeably for any data set in which a set of (not necessarily named) attributes can be identified. We consider three important data discovery problems: finding joinable tables (Section 2), finding unionable tables (Section 3), and creating an organization over collections of tables (Section 4). In the spirit of Open Data Integration, we are making our solutions and benchmark datasets, including a large collection of Canadian, US, and UK Open Data, open source.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2 Finding Joinable Tables

One of the basic ways of finding new tables is to start with an existing table (which we will call a query table) and look for other tables that join with the query table.

Example 2.1: A journalist is working on a piece about the state of industrial research in Canada. Being knowledgable about the topic, he searches using the keyword "NSERC"[1] on *open.canada.ca*, the Canadian Open Data portal, to find data sets related to the disbursement of government funding for scientific research. The keyword search returns tables on individual funding disbursements and the industry partners associated with some of the disbursements.[2] The search results are interesting, and include details on the funding amount, research topic, proposal keywords, along with other information. As one example, by joining and processing the tables on funding disbursements and industry partners, the journalist creates a table on companies and the total funding each company received from NSERC, in 2004, as shown in Table 1.

company name	total funding amount	number of grants	...
Bell Canada,	4,505,988	26	
Kruger Inc.	1,551,303	7	
Pfizer Canada Inc.	464,725	2	
Bombardier Inc.	120,000	1	

Table 1: The query table for finding joinable tables for attribute "company name"

The journalist is not yet satisfied because all the information in the table is related to the funding alone without information on the industry partners other than their names. He wants to look into other information about these companies that could potentially paint a more enlightening picture about why companies are receiving grant funding. In this case, the journalist does not know what specific information might be available and what search keywords to use. The keyword search feature on *open.canada.ca*, and other similar Open Data portals, only allows the user to search for keywords against the accompanying metadata, which is typically human-curated and may not contain many (or any) of the data values in the data set. So the journalist cannot just use the company names as the keywords to find tables that contain those company names.

The journalist wants to find *joinable tables*, other tables that join with the table he has already created on company names. So how do we build a search engine for joinable tables? The problem can be formulated as a classic *set overlap search* problem by considering attributes as sets, and joinable values as overlap between sets. For example, Table 1 has an attribute "company name" which contains the set of values: `Bell Canada`, `Kruger Inc.`, `Pfizer Canada Inc.`, etc. To find additional attributes about the companies the journalist wants to find tables that have an attribute that joins with these values.

2.1 Set Similarity Search

Set overlap search is an instance of the *set similarity search* problems that have been studied extensively [6, 13, 30, 43, 44]. However, these solutions focus on relatively small sets such as keywords, titles, or emails. The proposed algorithms typically use inverted indexes, and their runtimes are typically linear in the set sizes (both the query set and the sets being searched). However, Open Data has very large set sizes in comparison to keywords or titles: based on a crawl of 215,393 CSV tables from U.S., U.K., and Canadian Open Data portals as of March 2017, we found the average set size is 1,540 and the maximum size is 22,075,531. In comparison, typical data sets used in previous work for set similarity search [6, 44], have much smaller set sizes, under 100.

[1] Natural Sciences and Eng. Research Council of Canada is the federal funding agency for sciences (including CS) in Canada.

[2] *https://open.canada.ca/data/en/dataset/c1b0f627-8c29-427c-ab73-33968ad9176e*

Thus, the existing search algorithms that work for keyword and string search may not be suitable for finding joinable tables in Open Data.

Most of the previous work in set similarity search proposed algorithms that find exact results. In addition to exact approaches, some research has considered solving the set similarity search problem approximately. Interestingly, research in approximate approaches pre-date that of the exact approaches we mentioned above, and started outside of the database research community, in web search engines [9]. The most popular recipes for approximate set similarity search use Locality Sensitive Hashing (LSH) [4, 18, 23, 31, 41]. The reason for LSH's popularity is likely due to its simplicity: it is data-independent, requires only a few parameters, and can be implemented using off-the-shelf hash table code libraries. Most importantly, LSH also scales very well with respect to the size of sets, because each set is stored as a "sketch" – a small, fixed-size summary of the values (e.g., MinHash [8] or SimHash [11]).

The main justification for an approximate algorithm is that users are often willing to sacrifice a certain degree of accuracy in the search results to receive a significant speed-up in response time. Typically, an approximate algorithm returns much faster than an exact algorithm for the same query, and uses less resources (such as memory), because of the use of sketches instead of the actual data values. Thus, a search engine may respond quickly with approximate results first, and optionally provide the exact result only when required.

2.2 LSH Ensemble

We proposed LSH Ensemble, an LSH-based algorithm for approximate set overlap search, that can be used for finding joinable tables in Open Data [45]. Our work addresses two important research gaps in existing approximate set similarity search algorithms that were limiting their usefulness on Open Data. The first gap is in defining an appropriate similarity function. Most of the proposed LSH algorithms only support symmetric normalized similarity functions, such that $f(A, B) \rightarrow \mathbb{R}[0, 1]$, and $f(A, B) = f(B, A)$ regardless of the order of input. Examples of such similarity functions f are Jaccard and Cosine. The normalized version of set overlap is Containment similarity which in contrast is not symmetric:

$$Containment(Q, X) = \frac{|Q \cap X|}{|Q|} = \frac{Overlap(Q, X)}{|Q|} \rightarrow \mathbb{R}[0, 1] \tag{1}$$

where Q is the query set and X is a candidate set. Due to its asymmetry, containment similarity cannot be used by most existing LSH algorithms with the exception of Asymmetric Minwise Hashing (AMH) [41]. We used AMH on Open Data and observed that its accuracy deteriorates quickly when applied to data with a large skew in set cardinality size. We show the Zipfian distribution of attribute sizes in Open Data in Figure 1.

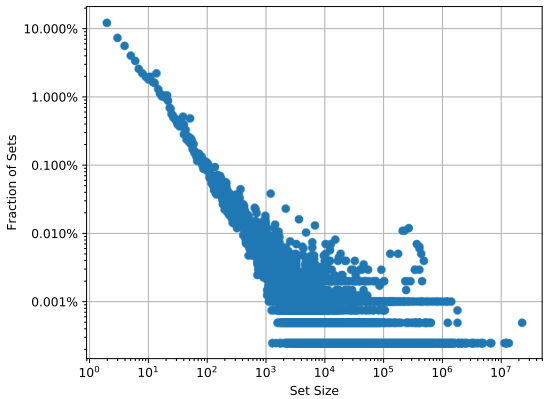


Figure 1: Set size distribution of sets from Canada, US, and UK Open Data

The large differences in set sizes is in fact the reason for choosing containment over other, better supported, similarity functions like Jaccard. When the query is large, the Jaccard similarity may be very small, even for sets that are fully contained in the query – sets we would want to return as fully joinable. The cardinality differences also leads to the second major gap in using LSH algorithms for searching Open Data: the accuracy of LSH decreases with skewness. AMH transforms Containment to Jaccard by making all sets the same size as the largest set. This approach pads small sets with added values that do not exist in any real sets and then builds sketches on the padded sets. We have shown that in Open Data where the largest set can be much larger than the smallest sets, AMH is very inaccurate.[3] An analysis for this effect is provided in the full version of our paper [46]. In Open Data, the set size distribution is close to Zipfian (with massive numbers of small sets), so the vast majority of sets will have very low accuracy sketches by using padding, and this is unacceptable even for approximate search. Our work addresses both of these gaps. We convert Containment to Jaccard, which is a normalized symmetric similarity function, using a well-known transformation [14].

$$Jaccard(Q, X) = \frac{|X \cap Q|}{|X \cup Q|} = \frac{Containment(Q, X)}{|X|/|Q| + 1 - Containment(Q, X)} \tag{2}$$

This makes it possible to adapt MinHash LSH using Jaccard to use Containment. However, the main problem is how to set the value of $|X|$, which is unknown at query time. Our approach uses an approximation, which goes hand-in-hand with reducing the effect of a skewed set size distribution that heavily impacts the accuracy of LSH algorithms. We partition all sets into disjoint buckets by their set size ranges – small sets will not be placed in the same buckets as large sets – and then build an LSH index for each bucket and use the smallest set size in each bucket as an approximation for $|X|$. The effect of partitioning is that as the cardinality difference within each bucket is smaller, the approximation is more accurate. Using this intuition, we developed an optimal partitioning strategy for Zipfian distributions of cardinality sizes [45]. Furthermore, since the query size $|Q|$ also affects the transformation from Containment to Jaccard similarity, and it is only given at run time, we use LSH Forest [4] so the LSH index can be tuned dynamically at run time for different query sizes.

2.3 Open Data Search Engine

Using LSH Ensemble, we built a search engine for Canadian Open Data [47], available at *demo.findopendata.com*.

Example 2.2: The journalist in our example could use this search engine to find tables that join with his query table in Table 1. As an example, he could find a table on contributions to political candidates[4] that lists every contribution made from companies to every political candidate from 2000 to 2004. By joining this table with the query table (and aggregating), the journalist could create Table 2.

company name	# of grants	total funding	total contribution
Bell Canada	26	4,505,988	22,385.22
Kruger Inc.	7	1,551,303	17,500
Pfizer Canada Inc.	2	464,725	11,100
Bombardier Inc.	1	120,000	5,000

Table 2: The result after joining the query table with a search result.

This resulting table is a product of very low effort by our journalist, who otherwise would rely on serendipity, significant knowledge of available data, or labor intensive search. Using our tool, the journalist unlocks a level of government transparency that was otherwise lost in the sea of Open Data.

[3] It is possible to over-compensate by using very large sketches, however, this just defeats the purpose of using sketches.

[4] https://open.canada.ca/data/en/dataset/af0b8907-8234-4fa3-b3be-f4406a9aacb8

3 Finding Unionable Tables

Some data analytics tasks require building master lists of tuples by unioning tables together. For instance, to analyze the state of greenhouse gas emission, a data scientist might need to build a list of various geographical locations and their gas emission indicators. A simple keyword search for "greenhouse gas emission" in the Canadian Open Data portal returns an overwhelming number of data sets which need to be manually examined to understand if the information can be meaningfully unioned. Automated solutions for more effectively finding unionable tables from Open Data would make this a less laborious task for data scientists.

Borough	Data Year	Fuel	ktCO2	Sector	...
Barnet	2015	Electricity	240.99	Domestic	
Brent	2013	Gas	164.44	Transport	
Camden	2014	Coal	134.90	Transport	
City of London	2015	Railways diesel	10.52	Domestic	

County	Year	Commodity Type	Total Emissions (MT CO2e)	Source	...
Benton	2015	Gasoline	64413	ConAgra Foods...	
Kittitas	2015	Fuel oil (1, 2...	12838	Central Wash...	
Grays Harbor	2015	Aviation fuels	1170393	Sierra Pacific...	
Skagit	2015	liquefied petroleum	59516	Linde Gas...	

Table 3: Unionable tables from Greenhouse Gas Emission of London (top) and Washington State (bottom).

3.1 Attribute Unionability

Defining when two attributes are unionable is not as straight-forward as defining when two attributes are joinable.

Example 3.1: Consider the data sets in Table 3 containing greenhouse gas emission statistics one from UK Open Data for the London area[5] and one from the US Open Data for the state of Washington[6]. Clearly, it seems reasonable to consider attributes with a large value overlap (like "Date Year" and "Year") to be *unionable*, but an important question is how much overlap is sufficient? In addition, value overlap is often not required. Attributes (like "Borough" and "County") that have semantic overlap may also be considered unionable. Sometimes an ontology containing both entities and classes can be used to discover that values that do not overlap actually come from the same semantic class (such as a class "Region of Country"). However, ontologies (especially public ontologies) are rarely complete and may have some, but not all values in an attribute. As with value overlap, an important question is how many values need to map to the same semantic class for two attributes to be declared unionable? Consider attributes "Fuel" and "Commodity Type" which are intuitively unionable. The former uses standard generic fuel types, while the latter uses user-defined text descriptions of fuels. It is possible, though unlikely, that a sufficient number of these values are in an ontology but even if we do not have an ontology with sufficient coverage, the words used in these two attributes have a semantic closeness that can be measured using natural language techniques. Intuitively, from our understanding of natural language, we recognize that the values in "Fuel" and "Commodity Type" appear frequently in the same context in natural language. Again, the challenge is to define a principled mathematical framework for justifying how close the words in two attributes need to be before the attributes are considered unionable.

This example illustrates some measures that can be used to quantify attribute unionability. Indeed, table union has been considered for web tables (meaning tables extracted from web pages [10]), using set overlap

[5] https://data.london.gov.uk/dataset/leggi

[6] https://catalog.data.gov/dataset/2015-greenhouse-gas-report-data

or other similarity functions over attribute values (Jaccard, tf-idf, etc.) or by mapping values to ontologies [10, 28, 40]. These heuristic approaches also leverage attribute names which can be very reliable in web tables. However, to really help our data scientist, we would like a principled way of turning a similarity measure into a hypothesis test: how likely is it that these two attributes contain values drawn from the same domain?

3.2 Table Unionability

Once we have a principled way of understanding how likely it is that two attributes are unionable, we need a way of evaluating how likely it is for two tables to be unionable.

Example 3.2: A data scientist who uses one of the data sets in Table 3 as a query table will receive in response a ranked list of tables that are unionable with her query by just using attribute unionability. However, if she uses a different query table, perhaps one containing only French words as values, over a corpus of English tables (where only an English ontology and word embeddings are used), it is not meaningful to return the nearest (closest) table if is too dissimilar to be unioned. So an attribute unionability measure alone is not enough. Moreover, if her query is a table like *London Energy and Greenhouse Gas Inventory* containing a subset of strongly unionable attributes with a candidate table *State of Washington Greenhouse Gas Report* in the repository (like "Date Year" with "Year", "Borough" with "County", and "Fuel" with "Commodity Type" of Table 3) as well as non-unionable attributes (like "Sector" and "Source"), we would want the search engine to return unionable tables with a maximal alignment indicating which attributes can be unioned.

From this example, we can add to our requirements for a search engine that it automatically selects which measure is most suitable for estimating the unionability of a specific pair of attributes. In addition, we would like a search engine that not only finds tables that are unionable on all their attributes, but rather one that retrieves tables unionable on a subset of attributes. To do this, we also need to be able to pick a maximal set of attribute pairs that can be meaningfully aligned (unioned) in two tables.

3.3 Table Union Search

Our solution estimates the likelihood that two attributes contain values that are drawn from the same domain - a problem we call *attribute unionability* [36]. We defined three types of domains for attributes and three statistical tests (*set-unionability* for overlap based on values, *sem-unionability* for overlap based on mappings into an ontology, and *NL-unionability* which is natural-language based). These tests evaluate the hypothesis that two attributes come from the same domain. Specifically, in set-unionability and sem-unionability, we argue that the size of the syntactic or semantic overlap of attributes follows a hypergeometric distribution. This allows us to compute the likelihood of two attributes being drawn from the same domain as the Cumulative Distribution Function of the hypergeometric distribution of their overlap. Given the cardinalities of A and B, n_a and n_b, the overlap size t, and the size of their domain n_D, the likelihood of set-unionability is defined as follows.

$$\mathbf{U}_{\text{set}}(A, B) = \sum_{0 \leq s \leq t} p(s | n_a, n_b, n_D) \tag{3}$$

To compute set-unionability, we need to know the size of the domain, n_D. In the context of Open Data, it is impractical or impossible to know the true domain. Thus, we make a practical assumption that the domain is approximately the disjoint union of A and B, and therefore $n_D \simeq n_a + n_b$. Similarly, we define the likelihood of the sem-unionability of A and B based on the size of overlap of the ontology classes to which their values are mapped.

In NL-unionability, we model an attribute with a multivariate normal distribution on the word embedding vectors ($\vec{v}$) of its values centered around μ_A with some covariance matrix Σ_A. To build such a distribution, we

leverage pre-trained word embedding vectors [25]. We call μ_A the *topic vector* of an attribute. Two attributes are NL-unionable if they are sampled from the same distribution. The distance between the topic vectors of attributes (the estimated mean of the set of embedding vectors) is an indicator of whether they are drawn from the same domain. We apply Hotelling's two-sample statistic (T^2) to undertake tests of the distances between the topic vectors of attributes. The distance of the mean of multi-variant distributions is inversely proportional to the T^2 statistic and follows an F-distribution. This allows us to define NL-unionability as the Cumulative Distribution Function ($\mathbf{F}$) of the distance between topic vectors.

$$\mathbf{U}_{\text{nl}}(A, B) = 1 - \mathbf{F}(T^2, n_a + n_b) \tag{4}$$

To automatically select the best measure for each pair of attributes in a repository, we also developed a new distribution-aware measure, *ensemble attribute unionability*. We defined the notion of *goodness* for a unionability score generated by each measure. Intuitively, given a measure and its corresponding unionability score for a query and a candidate attribute, goodness estimates the likelihood of the candidate being the most unionable attribute with the query in a given corpus. This allows the selection of the measure that provides the strongest signal, or goodness, within a repository. We developed LSH-based indexes that permit us to efficiently retrieve an approximate set of candidate unionable attributes. These indexes allow our search engine to scale to large repositories of Open Data.

To perform search, we need to define a unionability score for table pairs. Ideally, the most unionable tables share a large number of highly unionable attributes with a query table. However, in reality, candidate tables might share a large number of weakly unionable attributes or a few strongly unionable attributes with a query. We defined an *alignment* of two tables as a one-to-one mapping between subsets of their attributes. Assuming the independence of unionability of different attribute pairs in an alignment, we defined the probability of unionability of two tables as the product of their attribute unionability. Note that table unionability is not monotonic with respect to the number of attributes in a mapping in an alignment. Thus, we need to define a meaningful way of comparing the unionability between sets of attributes of different sizes. We presented the *goodness* of an alignment as the likelihood of the alignment being the most unionable of the alignments of a specific size in a repository. Through this distribution-aware algorithm, we are able to find the optimal number of attributes in two tables that can be unioned [36].

Since there is no benchmark available for table union search, to evaluate accuracy, we built and publicly shared an Open Data table union benchmark. Our experiments show that NL-unionability effectively finds unionable attributes, even when there is little value overlap and poor ontology coverage. In addition to greater precision and recall, NL-unionability has more pruning power than other measures. Also, ensemble unionability (our solution for choosing among unionability measures) improves the accuracy of table union search with only a small runtime cost [36]. We believe table union search enables cross-publisher analyses by letting data scientists quickly find data sets from different publishers that may have been gathered using similar protocols.

4 Data-driven Organization of Open Data

While the volume of Open Data continues to grow, Open Data repositories lack connectivity and a unified organization. Thus, Open Data tables are typically hosted in isolated silos, each with locally and manually curated organizations. Good publishing practice dictates that Open Data should be published with metadata. Typical metadata files contain unstructured or semi-structured information that might include the authoring institution, the date-time of the creation of the data set, and a set of *tags* or *keywords*. Open Data portals[7] currently use these tags for search and navigation. Since the metadata is manually created by data content authors, it is often incomplete, inconsistent, and may exhibit different levels of semantic granularity.

[7] https://open.canada.ca/en and https://ckan.org/

Example 4.1: Consider our journalist studying the state of industrial research in Canada from Example 2.1. He starts his research with *NSERC industry partner awards*,[8] government awards given to industry-university collaborations. The schema of the data set consists of university institutions, departments, names of the researchers, and industry company names. Here is an incomplete list of the tags of this data set in the accompanying metadata: `NSERC`, `award`, `fellowship`, `training`, `postgraduate`, `postdoctoral`, `university`, `college`, `innovation`, `industry`, `science`, `engineering`, `technology`. These tags represent different aspects of the data set (like `science` and `training`) and semantics at different levels of granularity (like `NSERC`, a specific award-agency, and `award`, a more abstract notion). We can use clustering techniques, based on similarity of tags, to create topics (and groups of related tables) to form an organization for the tables. For example, a table with tags *University and college; award and expenditures*[9] shares some similarity with the tags of the NSERC industry partner awards mentioned above. Interestingly, another similar table on academic grants and expenditures has only *one* tag: `Summary tables`. One would not be able to discover the connection between these tables using only metadata.

4.1 Connectivity (Linkage) Graph

We have presented data-driven methods to infer linkages (similarity based on joinability or unionability) between data sets. This allows users to search, discover, and combine data sets from different repositories. The Open Data search engine (Section 2.3) allows interactive navigation of repositories by join linkages [47]. During navigation, joinable data sets can be integrated and the result of a join can used to create new join linkages. While we have presented these as search problems, we can view the join and union graphs as a way of organizing tables and navigating table. Similarly, semantic similarity of metadata can be used to create a metadata linkage graph.

4.2 Hierarchical Organization

In addition to dynamic and query-driven connectivity discovery, a user may want a navigation platform that lets her view the entire repository abstractly and navigate progressively into parts of the repository of interest. While metadata tags alone may be insufficient to organize repositories, we believe these tags, together with other connectivity graphs (based on join and union), can be leveraged to create hierarchical organizations. Traditionally, taxonomy induction and ontology construction approaches extract labels (classes in ontologies) from text documents and use hand-constructed or automatically discovered textual patterns, as well as label co-occurrences to infer taxonomies on extracted classes [27, 42]. In the Open Data organization problem, a user wants to explore and navigate the semantic space of repositories and narrow down search to more specific topics to find desired data sets. Here, the challenge is to construct taxonomies over groups of data sets by leveraging metadata (which is mostly incomplete) and connectivity graphs.

5 Open Problems

Data discovery has been studied mostly within the enterprise, over corporate data lakes, or over web data (for example, web tables), with little research on Open Data repositories. We briefly review some of this work and highlight implications for Open Data management and open problems that remain.

5.1 Finding Connections between Data Sets

Keyword-based search. To perform keyword search on web tables, OCTOPUS applies document-style search on the content and context of web tables [10]. The result of search is then clustered into groups of unionable

[8] https://open.canada.ca/data/en/dataset/c1b0f627-8c29-427c-ab73-33968ad9176e

[9] https://open.canada.ca/data/en/dataset/66def988-48ca-4cc5-8e0b-19d9b73f6266

tables using common syntactic document similarity measures. Pimplikar and Sarawagi consider a query that is a set of keywords each describing an attribute in a desired table and present a search engine that finds tables whose schemas match the query [38]. Such keyword-based table search algorithms take time linear to the size of attribute values and have been shown to perform well on web tables. It is an open question if these techniques could be made to scale over Open Data. For example, the WDC Web Table Corpus is massive (over 80 Million tables), but the average table has 14 tuples and the largest has only 17,030 [29], in contrast, in the Open Data we have reported on in this paper, the average set size is 1,540 (meaning the distinct values in an attribute) and the maximum size is 22,075,531.

Attribute-based search. Existing work on the problem of finding linkage points among data sources on the web propose lexical analyzers and similarity functions accompanied by a set of search algorithms to discover joinable attributes [20]. As with our work, linkage point discovery does not require the existence of schemas, but it is unclear the performance over data with the massive sizes and skewed distributions of Open Data. Ilyas et al. [22] learn syntactic patterns of an attribute and use this to perform attribute search. To search repositories, they adapt MinHash LSH to use these patterns. The system has only been evaluated over repositories of thousands of attributes, but it is a very interesting open question to see if some of the ideas of LSH Ensemble [45] could be used to made this approach feasible on massive repositories, where individual attributes can also be massive.

Schema and table instance-based search. Data Civilizer, proposed after LSH Ensemble, also employs LSH to build a linkage graph between data sources in a data lake [12]. Extending ideas from enterprise data integration systems like Clio which compute possible join paths within data sources (relational or semi-structured databases) to suggest mappings between data sources [15], Data Civilizer computes possible join paths (foreign key paths) between data sources that can be used in queries [12]. Unlike our work which supports computation of linkages at interactive speed (the search problem), this work materializes a linkage graph. The system provides a declarative query language that enables search over this materialized graph [33]. Recent related work adds semantic linkages using word embeddings on attribute names in schemas (which are rich in enterprise data) [17] and dynamically maintains the materialized graph [16]. Other systems for data discovery over data lakes include Kayak [32] which also uses materialized statistical profiles to discover or suggest data sets.

Das Sarma et al. [40] define joinability and unionability for web tables by assuming that each table has a subject column that contains entities that the table is about. Their entity-complement approach uses an ontology to identify entities and the accuracy depends on the coverage of this ontology. We have shown that in Open Data the coverage of open ontologies like YAGO can be very low (on our corpus less than 13% of string values can be mapped to the ontology) [36].

A lot of research has been done on scalable schema matching and ontology alignment, where the problem is to match pairs (or small sets) of large schemas (ontologies). However, schema matching and ontology alignment have not generally been studied as search problems over massive collections of Open Data tables. Despite the rich literature, there are many open problems. Here we suggest a few. With the exception of set-unionability, the similarity measures used in table union work (ours and unioning of web tables) are not directly applicable to numerical values. Evaluating the unionability of numerical attributes requires first understanding what each attribute quantifies using which measurement unit. For instance, in Table 3, using the header names, attributes *Total Emissions (MT CO2e)* and *ktCO2* might be recognized as unionable and can be consolidated after unit conversion. To make sense of quantities in web tables, Ibrahim et al. [21] represent <measure, value, unit> triples over taxonomies of measures (e.g., physical and monetary). This representation can be augmented with distribution properties of an attribute. Once numerical attributes are canonicalized, we need a way of computing the likelihood of numerical attributes being drawn from the same domain and then we could incorporate this idea into table union search.

In table union search, we used pre-trained embedding models [25] (trained on Wikipedia documents) to compute NL-unionability. This allows NL-unionability to take advantage of embeddings trained on external sources of knowledge and results in high accuracy of search. However, these embeddings are limited to the vocabulary of the training corpus. Training word embeddings on text documents and knowledge bases has

been extensively studied [35, 37]. With the abundance of Open Data, training domain-specific and generic embedding models on Open Data using metadata, schema and constraints is an interesting research problem which has applications beyond table union search.

The VADA project considers source selection using detailed models of user preferences [1]. Source selection is a classical data integration (and information retrieval) problem that has been studied for decades. Relevant data sets (sources) are identified (typically using keyword search or exploration) and selection is done based on a model of user preferences [1] or models of the cost and/or value of using a source [39]. Source selection usually is a task following data discovery, but nonetheless something that will be important to consider over Open Data.

5.2 Data Set Organization and Management

Goods [19] organizes enterprise data sets by building data set profiles that leverage query logs and metadata. These profiles are exposed to users through portals that allow faceted search on the metadata of data sets. Open Data does not come with query logs but providing faceted search over Open Data is a desirable goal. Skluma is a system for organizing large amounts of data (organized in different file types) which extracts "deeply embedded" metadata, latent topics, relationships between data, and contextual metadata derived from related files [5]. Skluma has been used to organize a large climate data collection with over a half-million files. Skluma uses statistical learning to allow data to be discovered by its content and by discovered topics, something that is especially important for headerless-files which also appear in Open Data.

DataHub [7] provides efficient management of versioned data in a collaborative environment that has a community of users. In contrast, Open Data is published "out in the wild" typically without strict versioning. An interesting open problem is detecting near versions of data and providing an efficient way of on-boarding Open Data into a data collaboration platform like DataHub. DataHub supports IngestBase, a declarative data ingestion platform where enterprise users can create sophisticated data ingestion plans to ingest operational data from sensors or other external sources (financial feeds, social media, etc.) [24]. Open Data however requires a data-driven way of discovering good ingest plans as it is published in a way that basic elements (like the schema or number and type of attributes) may change.

6 Conclusion

We have taken a detailed look at data discovery over Open Data in a way that will be both efficient and effective given the characteristics of both individual data sets and collections of Open Data. We have presented new table join and table union search solutions that provide interactive search speed even over massive collections of millions of attributes with heavily skewed cardinality distributions. Through examples, we showcased how powerful search solutions make finding Open Data an easier, less daunting task – we hope making Open Data a bit more transparent and useable. Of course our solutions also inform enterprise data integration as more and more organizations are seeking to leverage external information including Open Data, and are creating data lakes with the massive scale and characteristics (for example, cardinality skew) already seen in Open Data [26].

Our methodology for studying the integration of Open Data is by developing data integration and preparation solutions that are themselves open. We are making our solutions open source, adding to the growing ecosystem of data preparation and integration tools.[10] We are also publishing curated benchmarks for data discovery problems (including gold standard solutions).[11] In data integration, open benchmarks (with gold standard solutions) have proven very valuable [2]. We believe both efforts can provide tremendous benefit to our community in advancing data integration research and the level of reproducibility of our field.

Acknowledgments This work was partially supported by NSERC.

[10] Including `https://github.com/ekzhu/datasketch` and `https://github.com/ekzhu/lshensemble`

[11] Including `https://github.com/RJMillerLab/table-union-search-benchmark`

References

[1] E. Abel, J. A. Keane, N. W. Paton, A. A. A. Fernandes, M. Koehler, N. Konstantinou, J. C. C. Ríos, N. A. Azuan, and S. M. Embury. User driven multi-criteria source selection. *Inf. Sci.*, 430:179–199, 2018.

[2] P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller. The iBench integration metadata generator. *PVLDB*, 9(3):108–119, 2015.

[3] J. Attard, F. Orlandi, S. Scerri, and S. Auer. A systematic review of open government data initiatives. *Government Information Quarterly*, 32(4):399–418, 2015.

[4] M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660, 2005.

[5] P. Beckman, T. J. Skluzacek, K. Chard, and I. T. Foster. Skluma: A statistical learning pipeline for taming unkempt data repositories. In *SSDM*, pages 41:1–41:4, 2017. System Demonstration.

[6] A. Behm, C. Li, and M. J. Carey. Answering approximate string queries on large data sets using external memory. In *ICDE*, pages 888–899, 2011.

[7] A. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. Elmore, S. Madden, & A. Parameswaran. Datahub: Collaborative data science and dataset version management at scale. In *CIDR*, 2015.

[8] A. Z. Broder. On the Resemblance and Containment of Documents. In *Compression and Complexity of Sequences*, pages 21–28, 1997.

[9] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.

[10] M. J. Cafarella, A. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, Aug. 2009.

[11] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[12] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *CIDR*, 2017.

[13] D. Deng, G. Li, J. Feng, and W. Li. Top-k string similarity search with edit-distance constraints. In *ICDE*, pages 925–936, 2013.

[14] S. Duan, A. Fokoue, O. Hassanzadeh, A. Kementsietsidis, K. Srinivas, and M. J. Ward. Instance-based matching of large ontologies using locality-sensitive hashing. In *ISWC*, pages 49–64, 2012.

[15] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos*, pages 198–236, 2009.

[16] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *ICDE*, 2018. To appear.

[17] R. C. Fernandez, E. Mansour, A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE*, 2018. To appear.

[18] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *PVLDB*, pages 518–529, 1999.

[19] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *SIGMOD*, pages 795–806, 2016.

[20] O. Hassanzadeh, K. Q. Pu, S. H. Yeganeh, R. J. Miller, L. Popa, M. A. Hernández, and H. Ho. Discovering linkage points over web data. *PVLDB*, 6(6):444–456, 2013.

[21] Y. Ibrahim, M. Riedewald, and G. Weikum. Making sense of entities and quantities in web tables. In *CIKM*, pages 1703–1712, 2016.

[22] A. Ilyas, J. M. F. da Trindade, R. C. Fernandez, and S. Madden. Extracting syntactic patterns from databases. *CoRR*, abs/1710.11528, 2017.

[23] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality.

In *STOC*, pages 604–613, 1998.
[24] A. Jindal, J. Quiané-Ruiz, and S. Madden. INGESTBASE: A declarative data ingestion system. *CoRR*, abs/1701.06093, 2017.
[25] A. Joulin, E. Grave, P. Bojanowski, & T. Mikolov. Bag of tricks for efficient text classification. *ACL*, 2017.
[26] E. Kandogan, M. Roth, P. M. Schwarz, J. Hui, I. Terrizzano, C. Christodoulakis, and R. J. Miller. Labbook: Metadata-driven social collaborative data analysis. In *IEEE Big Data*, pages 431–440, 2015.
[27] J. Lehmann and P. Hitzler. Concept learning in description logics using refinement operators. *Journal of Machine Learning*, 78–203(1), 2009.
[28] O. Lehmberg and C. Bizer. Stitching web tables for improving matching quality. *PVLDB*, 10(11):1502–1513, 2017.
[29] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer. A large public corpus of web tables containing time and context metadata. In *WWW*, pages 75–76, 2016.
[30] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE 2008*, pages 257–266, 2008.
[31] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007.
[32] A. Maccioni and R. Torlone. Crossing the finish line faster when paddling the data lake with kayak. *PVLDB*, 10(12):1853–1856, 2017. System Demonstration.
[33] E. Mansour, A. A. Qahtan, D. Deng, R. C. Fernandez, W. Tao, Z. Abedjan, A. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Building data civilizer pipelines with an advanced workflow engine. In *ICDE*, 2018. System Demonstration. To appear.
[34] J. Manyika, M. Chui, P. Groves, D. Farrell, S. V. Kuiken, and E. A. Doshi. Open data: Unlocking innovation and performance with liquid information. McKinsey Global Institute, 2013.
[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NISP*, pages 3111–3119, 2013.
[36] F. Nargesian, E. Zhu, K. Pu, & R. Miller. Table union search on open data. *PVLDB*, 11(7):813–825, 2018.
[37] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.
[38] R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. *PVLDB*, 5(10):908–919, 2012.
[39] T. Rekatsinas, A. Deshpande, X. L. Dong, L. Getoor, and D. Srivastava. Sourcesight: Enabling effective source selection. In *SIGMOD*, pages 2157–2160, 2016. System Demonstration.
[40] A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, pages 817–828, 2012.
[41] A. Shrivastava and P. Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *WWW*, pages 981–991, 2015.
[42] R. Snow, D. Jurafsky, and A. Y. Ng. Semantic taxonomy induction from heterogenous evidence. In *ACL*, pages 801–808, 2006.
[43] J. Wang, G. Li, D. Deng, Y. Zhang, and J. Feng. Two birds with one stone: An efficient hierarchical framework for top-k and threshold-based string similarity search. In *ICDE*, pages 519–530, 2015.
[44] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD*, pages 85–96, 2012.
[45] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. LSH ensemble: Internet-scale domain search. *PVLDB*, 9(12):1185–1196, 2016.
[46] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. LSH ensemble: Internet scale domain search. *CoRR*, abs/1603.07410, 2016.
[47] E. Zhu, K. Q. Pu, F. Nargesian, and R. J. Miller. Interactive navigation of open data linkages. *PVLDB*, 10(12):1837–1840, 2017. System Demonstration.

Big Data Integration for Product Specifications

Luciano Barbosa[1], Valter Crescenzi[2], Xin Luna Dong[3], Paolo Merialdo[2], Federico Piai[2],
Disheng Qiu[4], Yanyan Shen[5], Divesh Srivastava[6]
[1] Universidade Federal de Pernambuco luciano@cin.ufpe.br
[2] Roma Tre University {name.surname}@uniroma3.it
[3] Amazon lunadong@amazon.com
[4] Wanderio disheng@wanderio.com
[5] Shanghai Jiao Tong University shenyy@sjtu.edu.cn
[6] AT&T Labs – Research divesh@research.att.com

Abstract

The product domain contains valuable data for many important applications. Given the large and increasing number of sources that provide data about product specifications and the velocity as well as the variety with which such data are available, this domain represents a challenging scenario for developing and evaluating big data integration solutions. In this paper, we present the results of our efforts towards big data integration for product specifications. We present a pipeline that decomposes the problem into different tasks from source and data discovery, to extraction, data linkage, schema alignment and data fusion. Although we present the pipeline as a sequence of tasks, different configurations can be defined depending on the application goals.

1 Introduction

The product domain represents a challenging arena for studying big data solutions that aim at performing data integration at the web scale. An impressive number of product specification pages are available from thousands of websites, each page associated with a product, providing a detailed product description that typically includes technical and physical features, price, and customers' reviews. Integrating the data offered by these pages to create a comprehensive, unified description of each product represents a foundational step to enable valuable applications, such as, question answering, price comparison and data driven market analysis.

In this paper we describe an overview of our ongoing research activities to address the issue of integrating data extracted from product specification pages. We present some results that we have already achieved, the lessons that we have learned from our experience, open problems and future directions.

1.1 End-to-End Data Integration Pipeline

We refer to an end-to-end data integration pipeline for product specification pages that includes several tasks, as depicted in Figure 1. Every task adopts methods and techniques from databases, information retrieval and

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

machine learning. Every task produces output to feed the successive task in the pipeline, but intermediate results could find other compelling application scenarios as well. To this end, we advocate that the pipeline must include an empirical evaluation benchmark for every task.

In our vision, the information need is expressed by an input set of sample pages. We observe that products are typically organized in *categories*, and hence we expect that the sample pages refer to products from the categories of interest. Our approach is inspired by the Open Information Extraction [6] paradigm: the schema for the target data is not specified in advance, and the categories of the target products do not refer to a predefined product taxonomy, but they are rather inferred from data in the product pages of the input sample and in the product pages that are gathered from the Web along a progressive and iterative process.

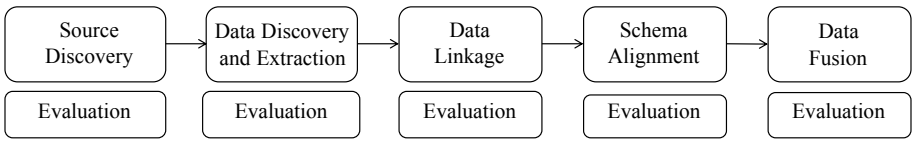


Figure 1: Our end-to-end big data integration pipeline for product specifications.

Source discovery aims at efficiently finding and crawling product websites in order to gather pages that refer to the products of interest. One might believe that discovering product websites is a minor task, as a relatively small number of *head sources* can offer enough data for most of the products. For example, amazon.com already provides data about an impressive number of products. However, valuable information is actually published by an enormous number of *tail sources* [3, 4], i.e., sources that each provide a small number of product entities. These tail sources are important because they improve coverage. They can often offer *tail entities*, i.e., products that are present in a small number of sources, as well as *tail attributes*, i.e., product properties that are present in a small number of entities. Also, tail sources ofter refer to *tail categories*, i.e., small niche categories of products. Finally, *tail sources* contribute to information diversity, as they provide values that depend on the local source, such as, product reviews and price. Source discovery also deals with efficiently crawling the discovered websites towards pages containing products of the categories of interest, without visiting unproductive regions.

Data discovery and extraction has the objective of processing the pages harvested in the previous task in order to locate and extract product attribute names and their values. As we mentioned above, we do not rely on a predefined schema, but rather extract attributes bottom-up, with the goal of discovering not just *head attributes*, but also *tail attributes* that cannot always be described in advance.

Data linkage seeks to cluster pages from different sources that refer to the same products. It is worth observing that in the traditional data integration pipeline, schema alignment is performed before record linkage [5]. Unfortunately, with a very large number of sources, such a traditional approach becomes infeasible because of the huge variety and heterogeneity among attributes. As we shall discuss later, we propose to perform data linkage before schema alignment as we can take advantage of the opportunity that products are named entities, and hence a product specification page usually publishes the product identifier.

Schema alignment addresses the challenge of semantic ambiguity and aims to reconcile the attributes offered by different sources, that is, to understand which attributes have the same meaning and which ones do not, as well as identify value transformations to normalize different representations of the same attribute values. Since we do not rely on a global schema given in advance, correspondences among attributes are established bottom-up leveraging the results of the previous data extraction and data linkage phases.

Data fusion tackles the issue of reconciling conflicting values that may occur for attributes from different sources. Data fusion aims at evaluating the trustworthiness of data, deciding the true value for each data item,

and the accuracy of the sources. To address these challenges, data fusion techniques rely on data redundancy, which further motivates the need to process many sources.

For simplicity of presentation, we have described our approach as a linear pipeline, where tasks are performed in sequence and independent of one another. However, there might be feedback loops between the tasks, as intermediate results can indeed influence the performance and the behavior of the preceding tasks and of the end to end solution.

1.2 Challenges and Opportunities

The web scale raises intriguing *challenges* for all the tasks of our pipeline due to its volume, variety, velocity, and veracity [5].

- The *volume* of data refers not only to the large number of products but, more importantly, to the number of sources. As we have discussed, to achieve coverage and diversity, we need to process a very large number of sources across the entire web, not just a small number of pre-selected sources.
- The *variety* of data is directly influenced by the number of sources and arises at many different levels, affecting every task of our pipeline. At the product category level, websites, especially the head ones, organize products according to such a vast plethora of categorization strategies that makes it very difficult, if not impossible, to reconcile them into a unified taxonomy. At the product description level, heterogeneities are related to attributes and values, which are published according to different granularities (e.g., physical dimensions in one field vs three separate fields for width, length, height), formats (e.g., centimeters vs inches) and representations (e.g., the color of a product as a feature vs distinct products for different colors).
- The *Velocity* of data involves the rate of appearance and disappearance of pages in sources as well as the rate of appearance and disappearance of web sources. Also, while some attributes (such as technical and physical features) are quite stable over time, the contents of the individual pages can change daily, for example for prices and reviews.
- The *Veracity* of data deals with honest mistakes that can occur in web pages, but also with deceits, that is, deliberate attempts to confuse or cheat (e.g., providing imprecise or erroneous product characteristics).

Our approach to address these challenges aims at taking advantage of the *opportunity* that products are named entities, and hence a product specification page usually publishes the product identifier. Web sources that deliver product specification pages publish product identifiers mainly for economic reasons: websites need to expose the product identifiers to let them be indexed by shopping agents and available to customers who search products for comparing prices or consulting specifications. Large e-commerce marketplaces strongly encourage sellers and retailers to publish product identifiers,[1] as they improve efficiency both for the internal management of data and for the exchange of data with search engines like Google and Bing.

The presence of identifiers allows us to drive the pipeline from source discovery to data integration by leveraging the opportunity of *redundancy of information at the global level*, and the *homogeneity of information at the local level*.

- At the global level, we observe that head (popular) products are present in several head (large) sources as well as in many tail (small) sources. Therefore, we expect that identifiers of head products are spread across many sources. Further, many head products in a category will often co-occur in multiple sources.

[1] eBay, Amazon, Google Shop explicitly require sellers to publish the id for many product categories. For example, see eBay's rules: `http://for-business.ebay.com/product-identifiers-what-they-are-and-why-they-are-important`.

- At the local level, we observe that the structure and the semantics of information, within each source, tend to be regular. Hence, we expect the product specification presented in a given page is published according to the same structure for every page in the same source.

Redundancy as a Friend Figure 2 illustrates the key intuitions underlying our approach [11] from source discovery to data integration to meet the goal of *effectively* and *efficiently* dealing with head and tail sources, hence including all pages of head and tail entities. Starting from known head entities in head sources, we take advantage of homogeneity of information at the local level to extract product specifications and identifiers for tail entities in head sources (even head sources offer many tail entities). Then, we exploit the presence of head entities across sources: searching head identifiers, we discover tail sources (even tail sources offer a few head entities). Again, we exploit homogeneity of information at the local level to extract identifiers and specifications for tail entities in tail sources.

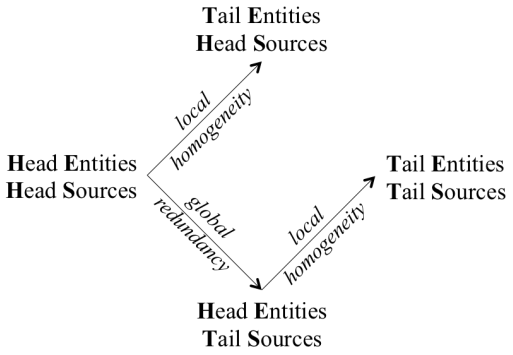


Figure 2: Our approach.

2 Source Discovery

We consider a focused crawler, Dexter [12], to discover and crawl product websites offering product pages for the input categories. The crawler iterates over three phases: *source finding*, *source crawling*, *identifier extraction*.

2.1 Source Finding

Our target websites are sparsely distributed on the web. To efficiently find them without visiting unproductive regions on the web, we consider two different strategies, *search* and *backlink*, whose results can be suitably combined. *Search* consists of querying a search engine with a set of seed identifiers of products; we expect that the search engine results contain pages from websites that publish information about these products. *Backlink* exploits services that provide inbound links, i.e., pages containing the set of links pointing to a given page; we rely on these services to find hubs, i.e., pages aggregating links to product websites.

These strategies allow us to discover many sources without penalizing recall, with Search focused more on head sources, Backlink including also tail sources. However, they often return also many non-relevant sites. To improve precision, we filter out results that are unlikely to represent a product website.

Search takes advantage of the redundancy of product information: searching the identifiers of known products on a search engine, we expect to find pages of the same products in many different sites. Identifiers to trigger the search engine are extracted from the input set of pages by means of suitable wrappers, or leveraging microdata annotations, such as schema.org, in case these are used in the pages.

Observe that the search engine can also return pages from websites that are not useful for our goals, such as pages from a web forum or pages from news websites. To efficiently select relevant websites, we search for multiple products and rank websites based on the number of pages from the same website that are present in the results. The rationale is that it is unlikely that a website that does not contain product specification pages appears frequently on the results of multiple queries with different product identifiers. Based on the ranking, we select the top-k websites.

It is worth observing that this strategy can penalize the recall of tail sources whose pages do not appear with a sufficient number of occurrences in the search engine results. This limitation can be further exacerbated by costs and limitations of the search engine APIs, which usually impose limits on the number of results per query, and on the number of queries in an interval of time. This is a significant issue if the goal is to collect an unbounded number of product websites.

Backlink aims to deal with the above limitations by adopting an alternate solution that is not dependent on the search engine restrictions. The idea is to find hubs, that is, pages that list links to many product websites. To this end we rely on online services that return the inbound links to a given input page.[2] Our approach is to search for pages that contain pages containing inbound links to the websites returned by Search. We then consider the domain of the links contained in these pages as candidate product websites. Similarly as for search engine results, backlinks can also lead to non-relevant hubs, subsequently leading to non-relevant websites. For example, sometimes they return generic hubs that point to multiple websites like popular websites in a country.

To select the most promising websites without penalizing the recall, we adopt an iterative approach to compute a ranking of the websites contained in the hubs returned by a seed set of product websites. As non-relevant hubs are less likely to point to many relevant websites, we score hubs based on the number of relevant websites they point to, and similarly we score the relevance of websites, based on the number of hubs that point to them. Based on this intuition, we rank websites and hubs and select the top-k websites.

Filtering Results The collection of websites discovered by Search and Backlink can contain many spurious results. In order to select our target websites, we adopt a simple machine learning approach, training a classifier to recognize if a website is a product website. The features that we have considered include all the anchor texts of the links in the home page.

2.2 Source Crawling

Source discovery also deals with crawling the discovered websites towards the product pages of the target categories. Also in this case, efficiency is a challenging objective, as websites often contain a huge number of pages, and only a small fraction may represent pages of interest.

We adopt a crawling strategy inspired by [8], which focused on web forums. Our approach builds on the assumption that, similar to forum websites, product websites have an internal structure consisting of one or more *entry pages* to the product content for a given category, followed by possibly paginated *index pages* that offer links to product pages. Therefore, our approach first looks for the entry pages related to the target category, then seeks for the index pages.

To discover the entry pages for a given category, the crawler relies on a classifier trained to predict links that are likely to point to entry pages. The classifier, which is trained with the links to entry pages of the target category in the websites of the input set of sample pages, analyses the anchors of links that are present in the home page and in target product pages (those returned by the search engine in the source discovery phase).

A similar approach, based on machine learning, has been adopted also to detect index pages. In this case, the classifier is trained to predict links that lead to product pages. A page is considered an index page if it contains a collection of links that point to product pages and that share uniform HTML format layout.

[2]These services are used to support search engine optimization (SEO).

2.3 Identifier Extraction

The last phase of source discovery has the objective of extracting product identifiers from the pages collected by the crawler. These identifiers will be used to launch new searches for discovering other sources.

To extract the product identifiers that can feed a new search, we exploit the local homogeneity of web sources. For each discovered source, we use the set of known identifiers to automatically annotate the pages collected by the crawler; then, for each annotation that exactly matches a textual leaf node of the DOM tree associated with the page, we infer an extraction rule (e.g. an XPath expression). To this end, we use the technique proposed by Dalvi *et al.* [2] to infer the extraction rules given noisy training data, that is, pages annotated by possibly erroneous labels. Because of the local structural homogeneity of the sources, we expect that the application of these rules returns the identifiers that are present in the pages.

3 Data Discovery and Extraction

We now describe our strategies to automatically extract product descriptions from the collections of product pages harvested by the previous task.

We have concentrated our efforts on specifications, in the form of pairs of attribute name and value, and on the product identifier. Future investigations will target our attention to extract and process also price and product reviews. We now describe our approach to extract specifications, and in Section 4 we illustrate our technique to extract product identifiers associated with the product on a page.

Automatically extracting specifications in many websites for different categories of products is challenging. Each website adopts a local template to generate the product specification pages. Then, to accurately extract the data, one should generate a specific wrapper for every website. A solution to this problem could be that of applying an automatic wrapper inference technique, such as Roadrunner [1]. However such a solution exhibits two major drawbacks: first, it has limited performance with irregular templates; second, it extracts a lot of meaningless data, since it considers every item that is not part of the template as data to extract. Another approach is to develop wrappers leveraging domain knowledge, as in [7]. However, as we already discussed, this would limit the opportunity of discovering *tail attributes*; also, with a large number of product categories this approach would require significant effort because of the heterogeneity of specifications across categories.

We have adopted a solution that represents a good trade-off between the above approaches. On the one hand, we exploit publication practices that occur globally in many product websites. On the other hand, we leverage the local homogeneity exhibited in large websites, which build pages according to a bunch of local templates.

Our solution splits the problem in two separate phases: *data discovery*, i.e., detecting the portion of product pages that contain the specifications, and *data extraction*, i.e., extraction of the specification, as attribute name and value pairs.

3.1 Data Discovery

We have observed that although specifications can be contained in different HTML structures, they are primarily found within HTML table and list elements. By manually inspecting 301 pages from a great variety of products, we have noticed that 62% of the specifications were inside an HTML table element, and 31% were inside an HTML list element; the remaining 7% were in other miscellaneous HTML elements.

Web pages may contain many tables and lists; however, we have noticed that, independent of product category and of the site, product pages exhibit structural characteristics that can be exploited to determine if a table or a list contains the product specifications (as opposed to being used, e.g., only for formatting purposes). Therefore, we have trained a classifier to detect tables and lists that contain product specifications. We used a Naive Bayes classifier, considering features dealing with table and list contents, such as, statistics about number of links, number of tokens in the leaf nodes, depth of the leaf nodes, etc. (see [12]).

3.2 Data Extraction

In order to extract attribute name and value pairs, we have considered two strategies.

The first strategy adopts a simple heuristic based on the observation that the structure of the fragments containing the specifications in tables and lists are very homogeneous, even across sources. By inspecting these tables and lists, we determined that attribute-value pairs of the specification are often contained in the html row element and that the first and second text elements of each row represent the attribute name and value, respectively. A similar heuristic is applied for the elements of lists classified as specifications.

The second strategy, which is applied on tables, uses the same technique adopted to extract the identifiers, described in Section 2.3. In this case, attribute names and values from the input sample pages are used to annotate the pages of the discovered sources. From the annotations we infer extraction rules. To obtain a more effective approach, we generalize these extraction rules to work on all the rows of the table, thus extracting all the attributes that it offers, including those that are not in the input set.

4 Data Linkage

Our approach to data linkage for product specification pages exploits the opportunity that product pages usually contain a product identifier. However locating the product identifiers in product pages at Web scale is quite challenging:

- It is not easy to locate, within the HTML of the product specification pages, the string that represents the identifier; some sources adopt microdata markups (such as schema.org), but their diffusion is limited [10]. Usually identifiers consist of a single token that, for some categories of products, follow specific patterns. But at Web scale, it is not possible to generalize these patterns (as done, for instance, in [9], which concentrated on a handful of sources), because of the large variety of patterns. Similarly, it is not possible to focus only on specific patterns, e.g., those associated with popular standards (as done, for instance, in [13]) because of the skewed distribution of the patterns. To give a concrete example, we have analyzed the identifiers extracted from the subset of pages annotated with microdata markups in the collection of sources discovered by our pipeline. We observed 930 different patterns for $33,281$ values, with the most frequent pattern (a sequence of 8 digits) matching less than 23% of values; the most frequent pattern associated to a standard was GTIN-13, with frequency 3%.

- Product pages usually contain identifiers not only for the main product presented in the page, but also for related products (e.g., suggested products, products bought by other users, sponsored products, etc.).

- Some identifiers may only be local (i.e., only within a source), not useful for linkage across sources. Local identifiers from different sources may conflict; similarly, conflicts may occur among global identifiers of products from different categories. Hence, different product pages associated with the same identifier are not guaranteed to refer to the same product.

To overcome these challenges, we leverage the redundancy of information that occurs at the global level and the regularities of the sources and uniqueness of information that occur at the individual source level [11]. Our approach to data linkage for product pages consists of an iterative phase to extract and filter identifiers, and a conclusive phase to resolve conflicting identifiers and generate the linkages.

4.1 Identifiers Extraction and Filtering

We start from a *seed set* of high quality product identifiers, which are used as keywords for searching among the discovered sources product pages that might refer to these products.

Next, for every retrieved product page, we infer an extraction rule for every HTML region containing an occurrence of the searched identifiers. To infer the extraction rule we use again the technique based on noisy annotations. However, here we consider as worth extracting also regions that contain the identifiers, not only those that exactly match the identifiers. In fact, for the purpose of linkage we are interested to extract the identifier that corresponds to the product on the page, and in many sources this is in a textual node, together with other tokens.

Based on the assumption of the local regularities of the sources, the extraction rules are used to obtain regions containing identifiers from all the other product pages in the same source. From every region returned by the rules, we have to select the token that represents a candidate identifier for the primary product of the corresponding page.

Since we cannot rely on a set of predefined patterns to select identifiers among the tokens of large regions, we consider that usually a source provides at most one page for a product. Therefore, we take the frequency of the tokens as a measure of their ability to serve as identifiers.

An immediate idea is to consider the token source frequency, that is, the token with the smallest number of occurrences within the source. However, considering every source separately does not work well when searching for global identifiers because many sources, especially tail sources, contain tokens that are locally rare, but do not represent global identifiers. For example, consider a small source where there is just one laptop with a touchscreen: if the keyword `touchscreen` is used along with the description of the product, it would be very rare at the source level, and thus it could be erroneously regarded as a good candidate identifier. Even if these tokens might appear as very selective at local level, they are much more frequent if considered globally, especially in the head sources. Continuing the above example, `touchscreen` is a pretty frequent token at the global level.

Therefore, we consider the token collection frequency, defined as the total number of occurrences of a token in the whole collection. In this way, we take into account both local and global characteristics. It is worth noting that because of this property we can perform data linkage only once we have gathered a large number of sources.

The above extraction and selection processes may produce incorrect identifiers, for many reasons: a wrong selection of the candidate identifiers; a region returned by an inaccurate extractor; the presence of regions containing identifiers that do not refer to the main product of the page (e.g., suggested products). To improve the precision, we consider the duality between good extractors and correct identifiers: an extractor is good if the identifiers of its regions are correct; similarly, an identifier is correct if it comes from a region returned by a good extractor.

The selected identifiers are then iteratively used to trigger another search on the source dataset.

4.2 Resolution of Conflicting Identifiers

Due to the variety of information across sources, and the presence of local and global identifiers, at the conclusion of the iterations different products could share the same identifier across sources. To improve the accuracy of linkage, we identify these conflicting identifiers by considering that every source consists of a homogeneous set of products: although the criteria that define the uniformity of the product categories are local, not global, with a large enough dataset it is likely that many pairs of products co-occur in many sources because they adopt similar (even if they are not identical) categories. Then, we consider identifiers that co-occur with multiple identifiers in many sources more reliable for the purpose of linking.

5 Lessons Learned and Future Work

In an experimental evaluation performed between Sept 2014 and Feb 2015, we have trained the focused Dexter crawler [12] to gather product pages from 10 coarse categories: camera, cutlery, headphone, monitor, notebook,

shoes, software, sunglasses, toilet accessories, televisions. The crawler discovered 3.5k websites, for a total of 1.9M pages. Each website contributed to provide pages for the different categories, and pages were grouped into $7,145$ clusters, corresponding to the local categories exposed by the websites (on average every websites has 2 local categories). The dataset is publicly available on-line.[3]

Building a Benchmark Product Dataset We compared the contents of our dataset with pages in Common Crawl,[4] an open repository of web crawl data. About 68% of the sources discovered by our approach were not present in Common Crawl. Only 20% of our sources contained fewer pages than the same sources in Common Crawl, and a very small fraction of the pages in these sources were product pages: on a sample set of 12 websites where Common Crawl presented more pages than in our dataset, we evaluated that only 0.8% of the pages were product pages.

These results suggest the critical need for the community to build a suitable benchmark product dataset to conduct big data research. Our dataset can be considered a first step in this direction. Another important step would be that of maintaining the dataset over time, as discussed next.

Maintaining the Benchmark Product Dataset: Addressing the Velocity Challenge In March 2018, we have checked all the URLs of the pages of the dataset. We have observed that just 30% of the original pages and 37% of the original sources are still valid (we consider a source valid if it contains at least one working URL). We also performed an extraction of the product specifications. We obtained complete specification from just 20% of the pages.

These numbers clearly indicate that the velocity dimension affects all the tasks of the pipeline. Developing solutions to collect snapshots over regular time intervals and perform data integration over time can open intriguing research directions. While some activities, such as checking the appearance/disappearance of sources can be done on monthly basis, others, such as crawling websites to check appearance/disappearance of pages and changes in the pages should be performed more frequently. To this end, the development of efficient incremental solutions for source discovery and web crawling represent interesting research directions.

As our experiments emphasize, data extraction rules are brittle over time. The development of wrappers resilient to changes in the pages has always been a primary goal in data extraction research. A dataset with multiple snapshots over a long interval of time, as the one that we have advocated above, could serve as a benchmark for data extraction solutions.

Harnessing Velocity We observe that while velocity represents a challenge, it could also become an opportunity. For example we observe that analyzing changes in the pages could help improve our data extraction and data linkage techniques. For example, examining the same product page over time may help us to more easily separate out information about related and suggested products, since those may change faster over time than the descriptions of the product in the page.

Schema Alignment We are currently working on the development of techniques to perform schema alignment for our product domain. The main difficulties that we have to face are due to the heterogeneity at the schema and at the instance level, due to the large number of independent sources.

To give a concrete example of the heterogeneity at the schema level, consider the dataset collected using the Dexter crawler, described earlier in this section. The specifications extracted from these sources contain more than $86k$ distinct attribute names (after normalization by lowercasing and removal of non alpha-numeric characters). Most of the attribute names (about $85k$) are present in less than 3% of the sources, while only 80 attribute names occur in 10% of the sources, with the most popular attribute name occurring in just 38% sources.

[3]Note that the on-line version (`https://github.com/disheng/DEXTER`) is an extension of the dataset presented in [12].

[4]`http://commoncrawl.org/`

The solution that we are investigating exploits data linkage to cluster attributes that share the same values. Since heterogeneity occurs also at the instance level, we aim at progressively finding correspondences between the clusters resolving increasingly complex matches between values with different representations.

Addressing the Veracity Challenge and Data Fusion Analyzing the sources of our dataset we noticed that some clusters contain product pages from different categories. In some cases, the errors are in the sources: some websites adopt unexpected (or even completely wrong) criteria as, for example, classifying monitors under a laptop category or vice-versa; other websites aggregate products and related accessories (which represent another category of products). In other cases, the errors are due to wrong classifications by the system.

To overcome this issues we want to investigate different solutions. First, we believe that introducing feedbacks in our pipeline could significantly improve data quality, especially for precision. For example, alternating source discovery and data linkage we could select better identifiers to feed to Search for source discovery, thus increasing the precision, with respect to the target category, of the sources. Similarly, results from schema alignment could help improve the precision of linkage, as pages whose attributes do not align are unlikely to represent the same product. Another promising direction to improve precision without penalizing recall is to study solutions to exploit humans in the loop. In particular, we aim at developing and evaluating techniques based on active learning and crowdsourcing to continuously train the classifiers with effective and updated training sets.

We have observed many inconsistencies between values of a product attribute across sources. Existing data fusion techniques [4, 5] can help to resolve these inconsistencies when they are due to honest mistakes, possibly in combination with extraction errors. However, the product domain also exhibits inconsistencies due to deceit, where sources may deliberately provide imprecise or erroneous product characteristics. Identifying and effectively addressing such inconsistencies at web scale is an important direction of future work.

Beyond Source Homogeneity Our approach to data extraction is based on the assumption that pages are structurally homogeneous at the local source level. This is a valid assumption for a vast majority of websites, but there are exceptions. For example, some websites that publish used products have a weak template and leave the seller the freedom to publish the product specification without any structural imposition. For some application scenarios, one can simply drop these sources. If on the contrary they are important, data extraction should be performed by solutions that do not rely on the template.

Beyond Product Specifications So far we have considered the extraction of the identifiers and of the specifications. However important data that complete the product description are price and reviews. Challenging issues for the extraction of price are to distinguish the price of the principal product in the page from the prices of other products, such as suggested product, similar products, and the actual price from discounts or list price. Reviews represent important information in many applications. An interesting problem is how to combine structured data from the specification with the unstructured data of the reviews.

References

[1] V. Crescenzi and P. Merialdo. Wrapper inference for ambiguous web pages. *Applied Artificial Intelligence*, 22(1-2):21–52, 2008.

[2] N. Dalvi, R. Kumar, and M. Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4(4):219–230, 2011.

[3] N. Dalvi, A. Machanavajjhala, and B. Pang. An analysis of structured data on the web. *Proceedings of the VLDB Endowment*, 5(7):680–691, 2012.

[4] X. L. Dong. How far are we from collecting the knowledge in the world? In *Keynote at 19th International Workshop on Web and Databases*. ACM, 2016.

[5] X. L. Dong and D. Srivastava. *Big data integration*, volume 7. Morgan & Claypool Publishers, 2015.

[6] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74, 2008.

[7] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang. Diadem: thousands of websites to a single database. *Proceedings of the VLDB Endowment*, 7(14):1845–1856, 2014.

[8] J. Jiang, X. Song, N. Yu, and C.-Y. Lin. Focus: learning to crawl web forums. *Knowledge and Data Engineering, IEEE Transactions on*, 25(6):1293–1306, 2013.

[9] H. Köpcke, A. Thor, S. Thomas, and E. Rahm. Tailoring entity resolution for matching product offers. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 545–550. ACM, 2012.

[10] R. Meusel, P. Petrovski, and C. Bizer. The webdatacommons microdata, rdfa and microformat dataset series. In *International Semantic Web Conference*, pages 277–292. Springer, 2014.

[11] D. Qiu, L. Barbosa, V. Crescenzi, P. Merialdo, and D. Srivastava. Big data linkage for product specification pages. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of data*. ACM, 2018.

[12] D. Qiu, L. Barbosa, X. L. Dong, Y. Shen, and D. Srivastava. Dexter: large-scale discovery and extraction of product specifications on the web. *Proceedings of the VLDB Endowment*, 8(13):2194–2205, 2015.

[13] A. Talaika, J. Biega, A. Amarilli, and F. M. Suchanek. Ibex: harvesting entities from the web using unique identifiers. In *Proceedings of the 18th International Workshop on Web and Databases*, pages 13–19. ACM, 2015.

Integrated Querying of SQL database data and S3 data in Amazon Redshift

Mengchu Cai, Martin Grund, Anurag Gupta, Fabian Nagel
Ippokratis Pandis, Yannis Papakonstantinou, Michalis Petropoulos
{mengchu, magrund, awgupta, fbnagel, ippo, yannip, mpetropo}
@amazon.com

Abstract

Amazon Redshift features integrated, in-place access to data residing in a relational database and to data residing in the Amazon S3 object storage. In this paper we discuss associated query planning and processing aspects. Redshift plays the role of the integration query processor, in addition to the usual processor of queries over Redshift tables. In particular, during query execution, every compute node of a Redshift cluster issues (sub)queries over S3 objects, employing a novel multi-tenant (sub)query execution layer, called Amazon Redshift Spectrum, and merges/joins the results in an streaming and parallel fashion. The Spectrum layer offers massive scalability, with independent scaling of storage and computation. Redshifts optimizer determines how to minimize the amount of data scanned by Spectrum, the amount of data communicated to Redshift and the number of Spectrum nodes to be used. In particular, Redshifts query processor dynamically prunes partitions and pushes subqueries to Spectrum, recognizing which objects are relevant and restricting the subqueries to a subset of SQL that is amenable to Spectrums massively scalable processing. Furthermore, Redshift employs novel dynamic optimization techniques in order to formulate the subqueries. One such technique is a variant of semijoin reduction, which is combined in Redshift with join-aggregation query rewritings. Other optimizations and rewritings relate to the memory footprint of query processing in Spectrum, and the SQL functionality that it is being supported by the Spectrum layer. The users of Redshift use the same SQL syntax to access scalar Redshift and external tables.

1 Introduction and Background

The database literature has described mediators (also named polystores) [6, 1, 4, 2, 3, 5] as systems that provide integrated access to multiple data sources, which are not only databases. In response to a client query that refers to multiple sources, the mediator formulates a distributed query plan, which obtains source data by issuing subqueries and/or (generally) data requests to the sources and consequently merges the results received from the multiple sources.

The advent of scalable object storage systems, such as Amazon's S3, has created a new kind of non-DBMS data source. The S3 data are stored across multiple storage nodes, which are accessible by the multiple *Compute*

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

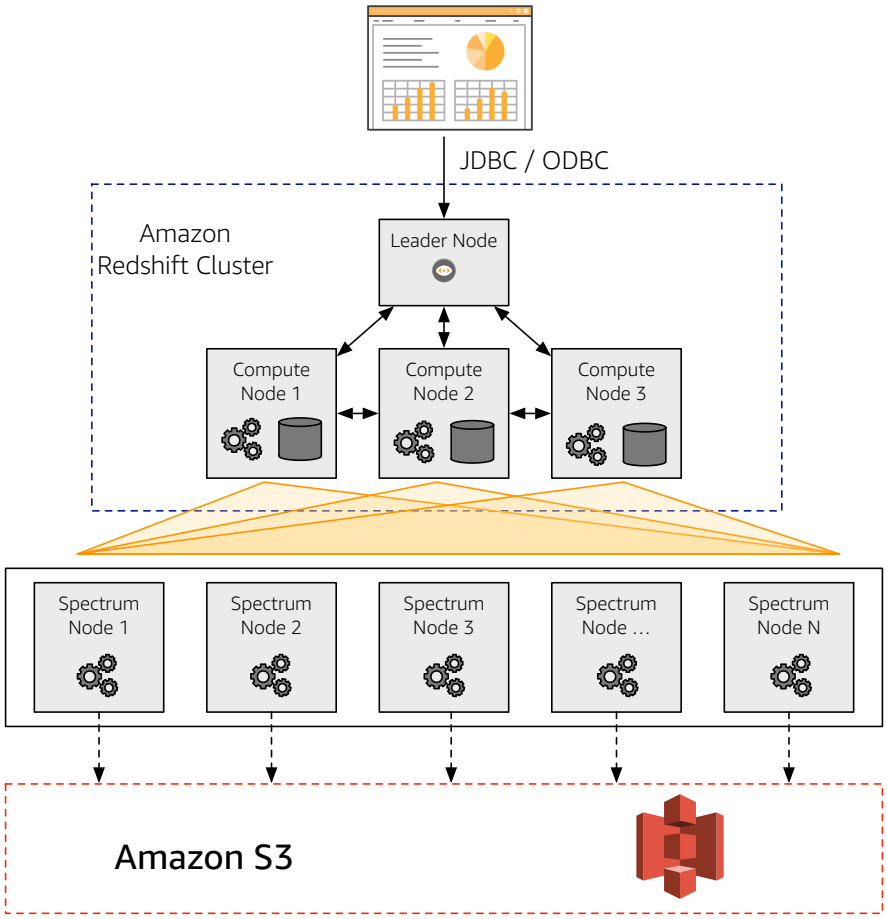


Figure 1: Amazon Redshift and the Spectrum processing layer

Nodes of a parallel database. Many users have data in an Amazon Redshift database and also have data in Amazon S3. A typical use case is very large fact data (in the data cube terminology) residing in S3, with matching dimension tables residing in Amazon Redshift. For such users, Amazon Redshift acts as mediator: It provides a logical view of the S3 data as *external tables* in addition to providing access to the Redshift tables. Queries received by Redshift may refer to both the Redshift tables and the S3 data, while the SQL syntax used when accessing scalar tables, regardless whether they are in Redshift or external, it remains the same. We discuss query planning and query processing issues solved by Redshift.

Using Spectrum in Query Processing. Spectrum is a multi-tenant execution layer of Amazon Redshift that provides massively scalable access and SQL processing capabilities over collections of Amazon S3 objects. It efficiently reads records from objects of various file formats stored in Amazon S3[1] and processes them before streaming the results to Redshift's Compute Nodes. The benefit of massive parallelism is realized when each Spectrum node computes a (sub)query that processes a lot of data but the results returned to Redshift are relatively small. Queries that filter, project and aggregate have this property. Thus, when accessing external tables on S3, Redshift's query optimizer pushes filters, projections and aggregations to the Spectrum layer, while joins (either between local and external tables or even between external tables), order-by's and final aggregations (of

[1]Currently, the supported file formats include columnar formats like Parquet, RCFile and ORC, as well as row-wise formats like CSV, TSV, Regex, Grok, Avro, Sequence, JSON and Ion.

the results of the multiple subqueries that have been sent to the Spectrum layer) are computed at the Redshift Compute Nodes. As we exhibit in Section 2, external table data filtering includes filters that correspond to semi-join reductions (Section 3.4). Furthermore, Redshift is aware (via catalog information) of the partitioning of an external table across collections of S3 objects. It utilizes the partitioning information to avoid issuing queries on irrelevant objects and it may even combine semijoin reduction with partitioning in order to issue the relevant (sub)query to each object (see Section 3.5).

Spectrum's massive parallelism comes with some challenges for the mediator/query planner: The Spectrum nodes are *stateless* - they are not supposed to cooperate and keep state in multi-phase plans. Second, the Spectrum subqueries must be computable within a memory budget; no local disk is used by Spectrum nodes. The hard memory limit leads to the use of partial aggregation (Section 3.3) and run-time decided semijoin reduction (Section 3.4). Further, the SQL functionality supported by each Spectrum node, is not fully aligned to the one of Redshift. Thus, the mediator (Redshift) also considers which SQL processing can be pushed to the Spectrum execution layer and which needs to be complemented in the Redshift processing layer.

2 Running Example

The setting of the scenario is an imaginary future where JK Rowling is launching the 8th book in the original Harry Potter series and the publisher's marketers want to direct the marketing campaign for this book. The particular marketers are in Miami and are looking to build a billboard advertising campaign. They had set such billboard campaigns in the past and they want to find out where and how much they succeeded. So, they want to find out the regions (postal codes) in Miami where the book sales were most improved by the past campaigns. The described scenario was exhibited in Amazon's reInvent 2017 conference, with an exabyte sales (fact) table.[2]

The first step to answer this question is to create a temporary table `hp_book_data` that holds the raw aggregated data about each Harry Potter book sales per Miami zip code, for the sales that followed within 7 days of its release. The second step of the analysis is to compare the book-over-book improvements per zip code and join with knowledge of which release/zip code combinations had billboard campaigns. Thus a comparison can be made between the cases that had the benefit of a campaign and those that did not. The second step is not challenging from a performance point of view, since `hp_book_data` will be very small. Rather the challenge is in computing `hp_book_data`.

The marketers have data in both the Redshift database and on S3. In particular, they have an exabyte fact table `S3.D_CUSTOMER_ORDER_ITEM_DETAILS` stored as a collection of S3 object, while the dimension tables are in Redshift. Having fact tables as external tables on S3 and dimension tables at Redshift table is indeed a common case of Redshift usage of external tables. In this case there are two tables, named `ASIN_ATTRIBUTES` and `PRODUCTS`, that provide information (author, title, release date) about the book editions, which are identified by the `ASIN` numbers. The `REGION` dimension table provides the country, state and city of the `REGION_IDs` that appear in the fact table.

The S3-residing fact table is partitioned by the `ORDER_DAY` attribute. The `partitions` table in the catalog provides the association between `ORDER_DAY`'s and the id's of the object prefixes (partitions) that are associated to each `ORDER_DAY`.

The following query computes the `hp_book_data`.

```
SELECT
   SUM(D.QUANTITY * D.OUR_PRICE) AS SALES,
   P.TITLE, R.POSTAL_CODE, P.RELEASE_DATE
FROM
  S3.D_CUSTOMER_ORDER_ITEM_DETAILS D,
  ASIN_ATTRIBUTES A, PRODUCTS P, REGIONS R
```

[2]The scenario is imaginary and, to the best of our knowledge, does not correspond to a particular marketing campaign.

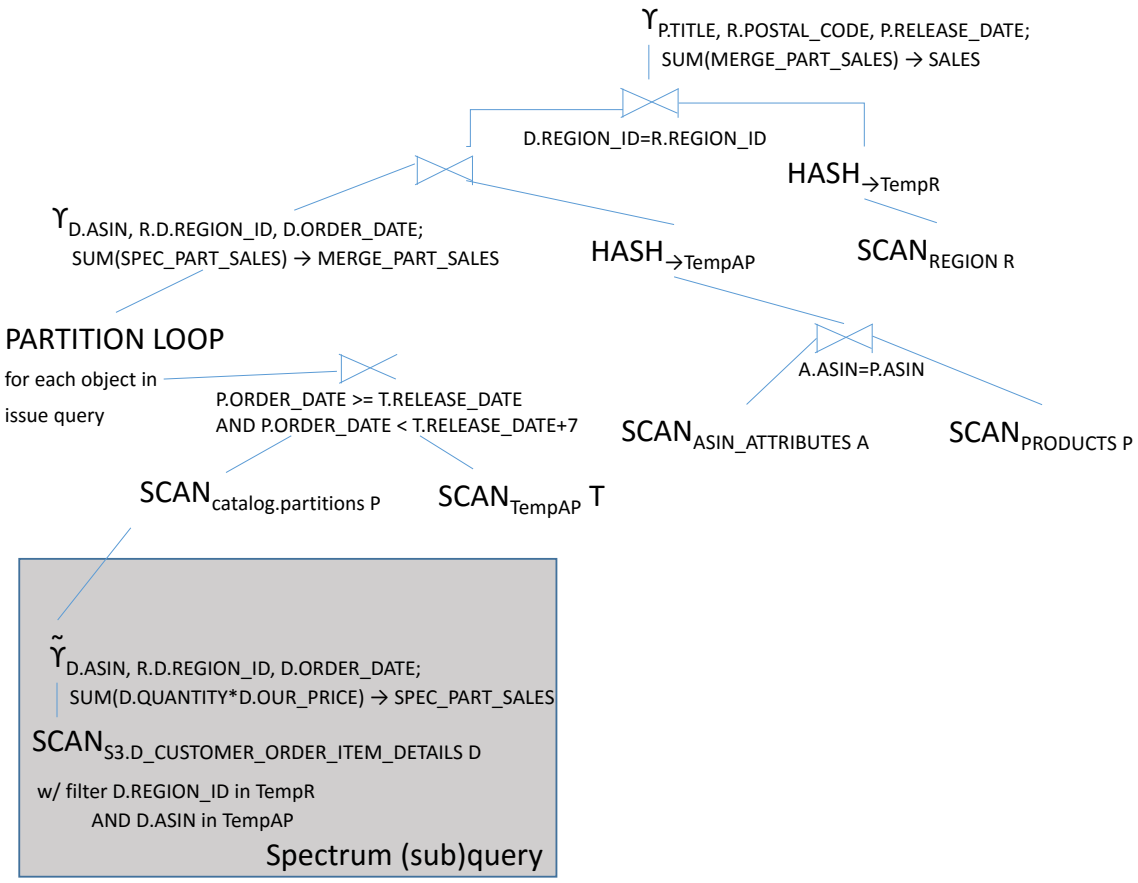


Figure 2: The query plan of the running example. The aggregation operator $\gamma_{\overline{G};f(A)\mapsto S}$ stands for a grouping on the grouping list $\overline{G}$ and aggregating with the aggregate funftion $f(A)$, leading to an aggregate attribute S. The partial aggregation variation $\tilde{\gamma}$ is explained in Section 3.3

```
WHERE
    D.ASIN = P.ASIN AND P.ASIN = A.ASIN AND
    D.REGION_ID = R.REGION_ID AND
    A.EDITION like '%FIRST%' AND P.TITLE like '%Potter%' AND
    P.AUTHOR = 'JK Rowling' AND
    D.ORDER_DAY >= P.RELEASE_DATE AND
    D.ORDER_DAY < P.RELEASE_DATE + 7 Days AND
    R.COUNTRY_CODE='US' AND R.STATE = 'WA' AND
    R.CITY = 'Miami'
GROUP BY
   P.TITLE, R.POSTAL_CODE, P.RELEASE_DATE
```

The plan of Figure 2 executes the query. Informally, its steps are:

1. Combine the Redshift tables `Products` and `ASIN_Attributes` to find the `ASIN`'s, `TITLE`'s and `RELEASE_DATE`'s of the first editions of Harry Potter books. There will be relatively few such tuples.

2. Scan the `REGIONS` table to find the `REGION_ID`'s and `POSTAL_CODE`'s of the Miami regions. Again,

there will be relatively few such tuples.

3. Retrieve from the catalog the list of partitions, identified by ORDER_DAY, that have sales data for the 7 days that followed each release. (See the semijoin in Figure 2.)

4. To each S3 object contained in a qualified partition, issue a Spectrum query that aggregates the revenue per Miami region, book edition and date. Notice the Spectrum query also includes IN filters, sof that only Miami data and Harry Potter book editions are aggregated.

5. On the Redshift side, merge the partially aggregated results per S3 object returned by the Spectrum layer.

6. For each group pull the corresponding values from the dimension tables, by performing the joins.

7. Perform the final aggregation.

The efficiency of the plan comes from Steps 3 and 4. In Step 3, Redshift prunes work at the partition level, and ends up having to only process the (relatively few) objects that follow the 7-days-post-release condition. Consequently, it sends much fewer (sub)queries to the Spectrum layer than a blind querying of all objects would lead to. In Step 4, the large amount of data per object boils down to returning only a few tuples, thanks to the IN filters and the presence of the aggregation. Of course, it matters that the Redshift optimizer first executed the joins of Steps 1 and 2, figures the relevant regions and book editions, and thus focused Steps 3 and Steps 4 to partitions and data in S3 objects that matter.

3 Redshift Dynamic Distributed Query Optimization

We discuss next the optimization steps that Redshift engages into, focusing primarily on special aspects of the optimization.

3.1 Join Ordering

In its first step, the Redshift query optimization creates a query plan, as it would have done even if the S3 table (or S3 tables in the general case) were database tables. In a cost-based fashion, using the statistics of the local and (external) S3 tables it creates the join order that yields the smallest intermediate results and minimizes the amount of data that are exchanged. Since the S3 tables are expected to be much larger than the Redshift tables, the typical plan will have the S3 table(s) at the left-most side of joins sequences. We will focus the rest of our optimization discussion to this pattern.

While join ordering is an expected functionality of databases, it is interesting to note that many SQL-on-Hadoop engines do not offer such. Often their SQL queries are not declarative but rather the join orders have an operational/execution order meaning as well.

3.2 Aggregation PushDown

Pushing the aggregations down to Spectrum has a large effect on performance and scalability, as it dramatically reduces the amount of data each Spectrum node needs to return to the Redshift cluster. In the running example, the aggregation pushdown rewriting leads to Spectrum queries that perform a *pre-aggregation* by grouping on ASIN and REGION_ID. [3] The pre-aggregation of Spectrum, leading to SPEC_PART_SALES is followed by a

[3] From a technical perspective, the ORDER_DATE is also included in the grouping attributes of the pre-aggregation. However, since the example's partitioning dictates that each S3 object has the same ORDER_DATE, the ORDER_DATE is not really accomplishing grouping and could also be eliminated.

merge aggregation that aggregates the Spectrum derived SPEC_PART_SALES into the MERGE_PART_SALES.[4] A final aggregation happens after the joins, since the grouping attributes of the pushed-down aggregation and the query's aggregation are not the same. The generalization of the rewriting behind the push-down of aggregation below a sequence of joins is:[5]

$$\gamma_{\overline{F.G},\overline{D_1.G},\ldots,\overline{D_n.G};agg(e(\overline{F.A}))\mapsto R}(\ldots(F \bowtie_{c_1} D_1)\ldots\bowtie_{c_n} D_n)$$
$$= \gamma_{\overline{F.G},\overline{D_1.G},\ldots,\overline{D_n.G};postagg(preR)\mapsto R}(\ldots(\gamma_{\overline{F.G},\overline{F/c_1\ldots c_n};preagg(e(\overline{F.A}))\mapsto preR}F) \bowtie_{c_1} D_1)\ldots\bowtie_{c_n} D_n)$$

The notation $\overline{Rel.Attr}$ stands for a list of attributes from the table Rel. In the running example, the "fact" table F is the S3 table, which joins twice with dimensions tables (or expressions involving join tables) before the aggregation, i.e., $n = 2$. Figure 2 shows the plan before the aggregation pushdown. The condition c_1 of the first join is `D.ASIN = P.ASIN AND D.ORDER_DAY >= P.RELEASE_DATE AND D.ORDER_DAY < P.RELEASE_DATE + 7 Days` and the condition c_2 of the second join is `D.REGION_ID = R.REGION_ID`. The operator $\gamma_{\overline{F.G},\overline{D_1.G},\ldots,\overline{D_n.G};agg(e(\overline{F.A}))\mapsto R}$ denotes grouping the input on the (concatenation of the) lists of columns $\overline{F.G}$ from the table F and columns $\overline{D_i.G}$ from the dimensions tables D_i. In the running example, $\overline{F.G}$ happens to be empty, i.e., the original plan has no aggregation on the S3 table attributes. The notation $\overline{F/c_1,\ldots,c_n}$ stands for the list of F attributes that appeared in the conditions $c_1,\ldots,c_n$. Informally, the rewriting says that a pre-aggregation can be pushed to F by replacing the dimension attributes in the grouping list with the fact table attributes that appeared in the joins. Notice that despite our use of the intuitive terms "fact table" and "dimension table", the rewriting does not pose foreign key/primary key constraints between the "facts" and "dimensions".

The rewriting requires that the aggregation agg is an associative aggregation, such as SUM, MIN, MAX, etc. The expression e is arbitrary. For each aggregation agg, there is a corresponding pre-aggregation $preagg$ and post-aggregation $postagg$. For example, if the aggregation is $\texttt{COUNT}(e(\ldots)) \mapsto R$ then the preaggregation is $\texttt{COUNT}(e(\ldots)) \mapsto preR$ and the post-aggregation is $\texttt{SUM}(preR) \mapsto R$. Similar pre-aggregation and post-aggregation applies to all associative aggregation operators, such as MIN, MAX, etc. Other aggregate functions such as AVG and STDEV can be emulated by combinations of the associative aggregations. As is well known, a few aggregation operators, such as the median, are neither associative, nor emulatable by associative ones.

There are many special cases that can lead to simpler, more effective rewritings. For example, in certain cases knowledge of the foreign key constraint allows elimination of the post-aggregation. In other cases, the semijoin reduction may render the join at Redshift unnecessary.

Before we proceed in the next optimizations, we raise the question of the rewriting's effectiveness and applicability in the case where the memory needed for grouping exceeds the available main memory.

[4]The merge aggregation is actually two aggregations: A first *local* aggregation by the computing nodes of Redshift and a second, *global* integration of the results of the local one. Depending on the expected cardinality of the aggregation, the global integration step may be executed by a single or more compute nodes. We do not depict this two-staging in Figure 2.

[5]Note, an alternate form of this rewriting would involve an aggregation and a single join, at a time. This would push an aggregation below the join and thus positioning it above the next join in the chain, which would trigger the next application of pushing aggregation below join. In the running example, under this variant we would have (a) a first aggregation at Spectrum on fact table attributes ASIN, ORDER_DAY and REGION_ID, as is the case in Figure 2 also, then (b) the join with the ASIN_ATTRIBUTES and PRODUCTS tables, again as is in Figure 2, (c) an aggregation on dimension table attributes TITLE, RELEASE_DATE and (still) fact table attribute REGION_ID (this is the point where the variant is different from Figure 2), (d) the join with the REGIONS table and (e) the final aggregation on the dimension attributes TITLE, POSTAL_CODE and RELEASE_DATE dictated by the query. At present time we elect the "single" rewriting option that detects the full tree of joins and pushes a single aggregation below it, as typically the very first aggregation (which happens on the Spectrum layer) is responsible for (vastly) most of the performance optimization.

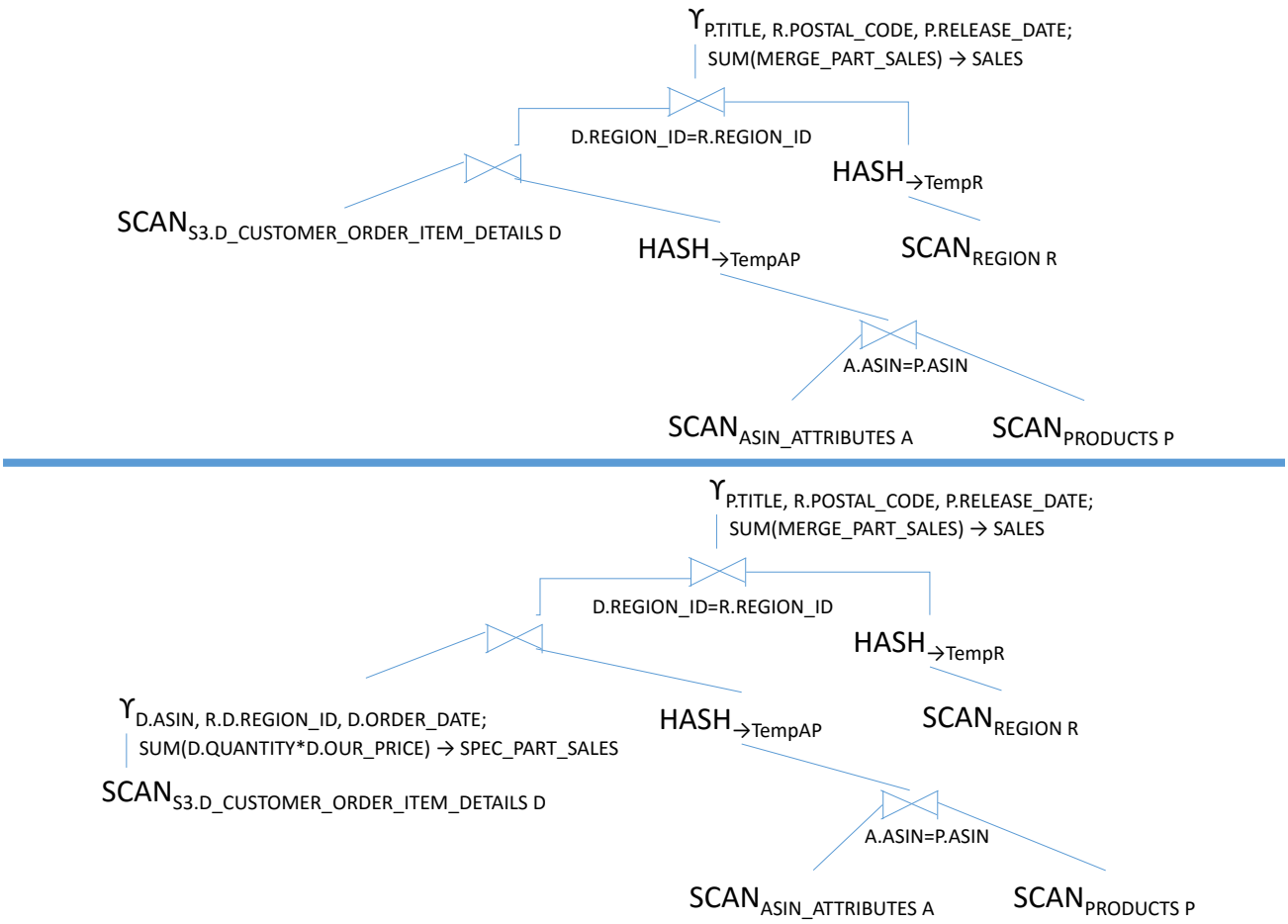


Figure 3: Plans before and after pushing aggregation below joins

3.3 Partial Aggregation

Aggregation pushdown comes with a risk: Aggregation, as usual, employs a data structure with pairs of group-by attribute values and aggregates. For each input tuple, the aggregation finds the relevant pair and updates the aggregate. If the group-by attributes of the input tuple have not appeared in any pair, then a new pair is created. But what if the grouping data structure exceeds the amount of memory that a Spectrum (sub)query may use? In the running example, this can happen if the amount of memory needed to store the pairs consisting of the group-by attributes `ASIN` and `REGION_ID` and the respective aggregate exceed the amount of available memory. This risk is mitigated by turning the pre-aggregation γ into a partial aggregation $\tilde{\gamma}$. A partial aggregation has generally a non-deterministic result and is allowed to output more than one tuples with the same grouping values. This allowance enables $\tilde{\gamma}$ to deal with aggregations that have results larger than the available memory, as follows. Assume that, at some point during the execution of $\tilde{\gamma}$ (a) the grouping structure has reached its maximum memory size and (b) a new input tuple arrives, with a `ASIN` a, `REGION_ID` r and $\langle a, r\rangle$ are not in the grouping structure. The $\tilde{\gamma}$ will pick a pair $(\langle a', r'\rangle, c')$ from the grouping structure, output it, and use its spot in the grouping table for the new entry $(\langle a, r\rangle, 1)$.[6] Multiple replacement strategies are possible. A good strategy

[6]Theoretically, $\tilde{\gamma}$ could output $(a, r, 1)$ and leave the grouping structure as-is. This would probably be the worst replacement strategy, since there is most probably locality in the arrival of tuples with the same `ASIN` and `REGION_ID`.

minimizes the probability that the newly evicted entry will need to be re-established in the grouping table, while keeping low the computation cost of discovering which entry to evict.

3.4 Semijoin Reduction by Dynamic Optimization

In the running example there are multiple regions but relatively few Miami regions. Similarly, there are multiple books but few Harry Potter books. Thus the Spectrum queries should focus on the few Miami regions and Harry Potter books. To achieve this, Redshift engages into semijoin reduction. Once the `REGION_ID`'s and `ASIN`'s are known (i.e., upon having evaluated the respective parts of the join tree), Redshift introduces the discovered `REGION_ID`'s and `ASIN`'s in the Spectrum queries, as `IN` *semijoin filter lists*. Thus a Spectrum query may look like (when formatted in SQL)

```
SELECT REGION_ID, ASIN, SUM(OUR_PRICE*QUANTITY) AS PART_SALES
FROM S3object
WHERE REGION_ID IN [45, 12, 179] AND ASIN IN [35, 6, 17]
GROUP BY REGION_ID, ASIN
```

There is a risk that the semijoin filter lists may be too long and exceed the available memory. Thus the semijoin reduction cannot be absolutely decided during query planning time. Rather, the Redshift query executor makes the decision at run time upon executing the filters and the joins between the dimension tables of Figure 2. If the right hand side results (`TempAP` and `TempR`) turn out to be small enough, then the queries issued to Spectrum will incude the corresponding semijoin filters.

3.5 Smart Partitioning, driven by Joins

The *partition loop* operator is responsible for finding the partitions that contain objects that are relevant to the query and emitting Spectrum queries to them. Its operation is based on the `catalog.partitions` table, which associates each partition of an S3 table with a value of the partition attribute(s) of the S3 table - the `ORDER_DAY` in the example. The partition attribute, which is typically time-related, is often constrained by the query. The simple form of constraining is when a selection filter applies on the partition attribute. For example, this would be the case if the running query also had a condition `D.ORDER_DATE > '04/01/2005'`. Any filter on the partitioned attribute should turn into a filter that is used to find the relevant partitions.

A more interesting case emerges when the constraint on the partition attribute is expressed by a join. In the running query, the `ORDER_DAY`s are constrained by the conditions placed on `PRODUCTS` and `ASIN_ATTRIBUTES`: The only `ORDER_DAY`s that matter are the seven days that followed a Harry Potter book release. Technically, these are the `ORDER_DAY`s in the result `TempAP` in Figure 2. Thus, only the partitions that are associated with these `ORDER_DAY`s should be queried. The *partition loop operator* (Figure 2) finds the partition id's by executing the depicted semijoin of the `catalog.partitions` with `TempAP`. Generally, the semijoin's condition is derived as follows: Detect the join conditions on the partitioned S3 table that involve the partition attribute. Assuming the join condition is in conjunctive normal form, pick the maximum number of conjunction arguments that involve the partition attribute and adjust the partition attribute references to refer to the catalog table (as opposed to the partitioned table).

4 Conclusions

Amazon Redshift provides integrated access to relational tables and S3 objects. The S3 data are accessed via a highly parallel, multi-tenant processing layer, called Amazon Redshift Spectrum. Multiple optimizations ensure that the queries executed at the scalable Spectrum layer process only the relevant S3 objects and return to the Redshift cluster small results, while cost-based optimizations, such as join ordering, are still in effect.

References

[1] F. Bugiotti, D. Bursztyn, A. Deutsch, I. Ileana, and I. Manolescu. Invisible glue: Scalable self-tunning multi-stores. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[2] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards heterogeneous multimedia information systems: The garlic approach. In *Proceedings RIDE-DOM*, pages 124–131, 1995.

[3] S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *IPSJ*, pages 7–18, 1994.

[4] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. B. Zdonik. The bigdawg polystore system. *SIGMOD Record*, 44(2):11–16, 2015.

[5] J. LeFevre, J. Sankaranarayanan, H. Hacigümüs, J. Tatemura, N. Polyzotis, and M. J. Carey. MISO: souping up big data query processing with a multistore system. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1591–1602, 2014.

[6] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

Robust Entity Resolution Using a CrowdOracle

Donatella Firmani[1], Sainyam Galhotra[2], Barna Saha[2], Divesh Srivastava[3]
[1] Roma Tre University, donatella.firmani@uniroma3.it
[2] UMass Amherst, {sainyam, barna}@cs.umass.edu
[3] AT&T Labs – Research, divesh@research.att.com

Abstract

Entity resolution (ER) seeks to identify which records in a data set refer to the same real-world entity. Given the diversity of ways in which entities can be represented, ER is a challenging task for automated strategies, but relatively easier for expert humans. We abstract the knowledge of experts with the notion of a boolean oracle, that can answer questions of the form "do records u and v refer to the same entity?", and formally address the problem of maximizing progressive recall and F-measure in an online setting.

1 Introduction

Humans naturally represent information about real-world entities in very diverse ways. Entity resolution (ER) seeks to identify which records in a data set refer to the same underlying real-world entity [4, 8]. ER is an intricate problem. For example, collecting profiles of people and businesses, or specifications of products and services from websites and social media sites can result in billions of records that need to be resolved. Furthermore, these entities are represented in a wide variety of ways that humans can match and distinguish based on domain knowledge, but would be challenging for automated strategies. For these reasons, many frameworks have been developed to leverage humans for performing entity resolution tasks [17, 9].

The problem of designing human-machine ER strategies in a formal framework was studied by [18, 16, 6]. These works introduce the notion of an *Oracle* that *correctly* answers questions of the form "do records u and v refer to the same entity?", showing how different ER logics can achieve different performance having access to such a "virtual" tool and a set of machine-generated pairwise matching probabilities. In this setting, the knowledge of experts is abstracted with the notion of a boolean oracle. However, certain questions can be difficult to answer correctly even for humans experts. To this end, the work in [7] formalizes a *robust* version of the above ER problem, based on a "Noisy oracle" that can *incorrectly* label some queried matching and non-matching pairs. The same paper describes a general error correction tool, based on a formal way for selecting indirect "control queries", that can be plugged into any correction-less oracle strategy while preserving the orignal ER logic. We refer to both the perfect and the noisy boolean oracle models as *CrowdOracle*.

Earlier CrowdOracle strategies, such as [18], consider ER to be an *off-line* task that needs to be completed before results can be used. Since it can be extremely expensive in resolving billions of records, more recent strategies [6, 7] focus on an *on-line* view of ER, which enables more complete results in the event of early

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

termination or if there is limited resolution time available. On-line strategies consider *progressive* recall and F-measure as the metrics to be maximized in this setting. If one plots a curve of recall (or, analogously, F-measure) as a function of the number of oracle queries, progressive recall is quantified as the area under this curve.

Contributions and outline We describe the CrowdOracle pipelines introduced by [6, 7, 16, 18] by using a common framework. The building blocks of the framework enable the definition of *new* strategies, which can perform even better than those in the most recent works in certain applications. Problem formulation, error correction and some examples are taken from [6, 7]. However, framework formulation, strategy categorization, and an illustrative experiment are original contributions of this paper. This paper is organized as follows.

- In Section 2, we describe our ER problem leveraging the formal notion of a CrowdOracle that can (possibly incorrectly) label some queried matching and non-matching pairs.
- In Section 3, we discuss the components of our descriptive framework for CrowdOracle strategies. Key techniques and theoretical results of [7] are also given in this section.
- In Section 4, we describe previous CrowdOracle strategies leveraging our framework, and show how combining its building blocks in new ways can lead to more efficient algorithms for specific applications.
- Finally, related work is discussed in Section 5.

2 Preliminaries

Let $V = \{v_1, \ldots, v_n\}$ be a set of n records. Given $u, v \in V$, we say that u *matches* v when they refer to the same real-world entity. Let $H = (V, A, p_m)$, $A \subseteq V \times V$, be a graph with pairwise machine-generated matching probabilities $p_m : A \to [0, 1]$. We may not have probabilities of all record pairs, and we may have $|A| << \binom{n}{2}$. Consider a graph $C = (V, E^+)$, where E^+ is a subset of $V \times V$ and $(u, v) \in E^+$ represents that u matches with v. C is transitively closed, that is, it partitions V into cliques representing distinct entities. We call the nodes in each clique a *cluster* of V, and we refer to the clustering C as the *ground truth* for the ER problem. We refer to the cluster including a given node u, as $c(u) \in C$. Consider a black box which can answer questions of the form "are u and v matching?". Edges in C can be either asked to the black box or inferred leveraging previous answers. If the black box always tells the truth, a user can reconstruct C exactly with a reasonable number of queries [18, 16]. In real crowdsourcing applications, however, some answers can be erroneous and we can only build a noisy version of C, which we refer to as C'. $c'(u)$ refers to the cluster in C' including a given node u.

Definition 1: A **CrowdOracle** for C is a function $q : V \times V \to \{YES, NO\} \times [0, 0.5]$. If $q(u, v) = (a, e)$, with $a \in \{YES, NO\}$ and $e \in [0, 0.5]$, then $Pr[(u, v) \in E^+] = 1 - e$ if a=YES, and e otherwise. In the ideal case, when $e = 0$ for any pair (u, v), we refer to the CrowdOracle as **perfect oracle**.

For instance, if $q(u, v) = (\text{YES}, 0.15)$, then $(u, v) \in E^+$ with probability 0.85, and if $q(u, v) = (\text{NO}, 0.21)$, then probability of $(u, v) \in E^+$ is 0.21. We refer to the probability of a specific answer for the pair (u, v) being erroneous, conditioned on the answer being YES or NO, as its *error probability* $p_e(u, v)$. Let $Q = Q_+ \cup Q_-$ be a graph containing all the edges that have been queried until a given moment, along with the oracle answers, we state $p_e : Q \to [0, 0.5]$. An ER strategy s takes as input matching probability graph H and grows a clustering C' by asking edges as queries to the noisy oracle. We call *inference* the process of building a clustering C' from Q. C' initially consists of singleton clusters: s can either merge existing clusters into larger clusters, or split an already established cluster. Note that the sub-graph of Q_- induced by $c'(u)$ (that is, $Q_- \cap c'(u)$) can be non-empty, because of wrong answers. We refer to such a sub-graph as $Q_-[c'(u)]$.

Input data There are many ways of estimating the *matching* probability function p_m. For instance, automated classifier methods can provide pairwise similarities, which can be mapped to matching probabilities, as in Section 3.1 of [20]. Analogously, there are many ways of accessing *error* probabilities. For instance, the crowd platform could return a confidence score associated with each answer. Another option is to learn a function mapping similarity scores to error probabilities, akin to matching probabilities [20]. However, computing all the $\binom{n}{2}$ pairwise matching probabilities may not be feasible when the number of records n is large, and adopting a crowd-only approach may be prohibitively expensive. To this end, people often remove obvious non-matching pairs during a pre-processing phase. Then, they ask the crowd to examine the remaining pairs, which lead to a relatively sparse graph. One approach is to put obviously non-matching nodes (e.g., watches and dishwashers in an e-commerce dataset) in separate "domains" and consistently remove cross-domain edges. The result, similarly to what is done in [15], is a collection of disconnected complete sub-graphs that can be resolved independently. Another approach, exploited for instance in [3, 14], is to remove obviously non-matching edges either by a matching probability threshold or other cheap procedures such as (overlapping) *blocking*. The result is a sparse graph, possibly consisting of several connected components (not necessarily cliques). In the traditional (non-crowdsourcing) setting, there is an extensive literature about blocking (see for instance [22]).

3 CrowdOracle Framework

We now define a conceptual framework for describing recent CrowdOracle strategies in terms of basic operations. The input of each CrowdOracle strategy includes the CrowdOracle answers Q, the matching probability function p_m, and the error probabilities p_e, as in definition of Problem 1. In our framework, a *strategy* is a mechanism for selecting non-exhaustive *sequence* of CrowdOracle queries (i.e., less than $\binom{n}{2}$ queries) and inferring clusters according to answers. An oracle strategy also uses the following shared data and methods.

- The partition C', which can be updated upon the arrival of new answers.
- The method `query_pair`(u, v), which returns a $\{YES, NO\}$ oracle answer for the pair (u, v). Every invocation of such method contributes to the cost of the ER process and each strategy can be thought of determining a different sequence of `query_pair`() invocations. We note that given two partially grown clusters $c_1, c_2 \in C'$ in the perfect oracle setting, the result of `query_pair`(u, v) is consistently the same for any $(u, v) \in c_1 \times c_2$. For sake of simplicity, then, we sometimes use notations such as `query_pair`(u, c_2) or `query_pair`(c_1, c_2) for referring to an arbitrary inter-cluster pair-wise query.

Our framework consists of three aspects, useful for describing a variety of CrowdOracle strategies:

- the cost model of the ER task, which can represent off-line or on-line ER;
- the CrowdOracle model and the selection criteria for issued queries;
- the algorithms for updating the partition C', which we refer to as "building blocks".

We discuss each of the above components in the following sub-sections.

3.1 Cost model

Recall and F-measure denote, respectively, the fraction of positive edges found among those of the unknown clustering C, and the harmonic mean of this value and *precision*, which is the fraction of positive edges found that truly belong to C. Specifically, F-measure is defined as $\frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$. Recall and F-measure naturally represent the "amount" of the information that is available to the user at a given point. However, they cannot distinguish the *dynamic* behaviour of different strategies. In other words, they cannot represent whether a strategy achieves

high F-measure only *at the end* of the ER process or *earlier on*, thus enabling early usage by the user. Consider the following example with perfect oracle access. In this ideal setting, we can leverage transitivity: if u matches with v, and v matches with w, then we can deduce that u matches with w without needing to ask the oracle. Similarly, if u matches with v, and v is non-matching with w, then u is non-matching with w.

Example 1 (Running Example): There are many colleges and universities around the Italian city of Rome, each with its own *aliases* or abbreviated names and acronyms[1] which humans can distinguish using domain knowledge. Given the six institutions (r_a) Università di Roma, (r_b) La Sapienza, (r_c) Uniroma 1, (r_d) Roma Tre, (r_e) Università degli Studi Roma Tre, (r_f) American University of Rome, humans can determine that these correspond to three entities: r_a, r_b, and r_c refer to one entity, r_d and r_e refer to a second entity, and r_f refers to a third entity. Let us assume that we have the following probability estimates for record pairs. Matching pairs: $p(r_d, r_e) = 0.80$, $p(r_b, r_c) = 0.60$, $p(r_a, r_c) = 0.54$, $p(r_a, r_b) = 0.46$. Non-matching pairs: $p(r_a, r_d) = 0.84$, $p(r_d, r_f) = 0.81$, $p(r_c, r_e) = 0.72$, $p(r_a, r_e) = 0.65$, $p(r_c, r_d) = 0.59$, $p(r_e, r_f) = 0.59$, $p(r_a, r_f) = 0.55$, $p(r_b, r_d) = 0.51$, $p(r_b, r_e) = 0.46$, $p(r_c, r_f) = 0.45$, $p(r_b, r_f) = 0.29$. We let some non-matching pairs have higher probability than matching pairs. Some observers, for instance, may consider r_a and r_e more likely to be the same entity, than r_a and r_b. Consider two strategies S_1 and S_2. Let S_1 ask subsequently (r_a, r_b), (r_a, r_c), (r_d, r_e), (r_a, r_d), (r_a, r_f), and (r_d, r_f). Let S_2 ask instead (r_a, r_d), (r_a, r_f), (r_d, r_f), (r_d, r_e), (r_a, r_b), and (r_a, r_c). Both strategies issue to the oracle the same 6 pair-wise queries and can get recall 1 by leveraging transitivity. However, the recall of S_1 would be 0.75 after labeling the first two record pairs and 1.0 after the third, while the recall of S_2 would be still 0.0 after the first three record pairs, 0.25 after labeling the fourth pair, 0.5 after labeling the fifth record pair, and 1.0 only after labeling the sixth record pair.

In response to the above concerns, we introduce two variants of recall and F-measure, dubbed *progressive recall* and *progressive F-measure*. If one plots a curve of recall as a function of the number of oracle queries, progressive recall denotes the *area* under the curve. Progressive F-measure is defined analogously.

CrowdOracle problems We are now ready to define our progressive CrowdOracle problems, where we aim for high F-measure early on, which we refer to as *on-line* ER, and its traditional *off-line* version. In the perfect oracle setting, optimizing F-measure is the same as optimizing recall.[2]

Problem 1 (On-line): Given a set of records V, a CrowdOracle access to C, and a matching probability function p_m (possibly defined on a subset of $V \times V$), find the strategy that maximizes progressive F-measure.

Problem 2 (Off-line): Given a set of records V, a CrowdOracle access to C, and a matching probability function p_m (possibly defined on a subset of $V \times V$), find the strategy that maximizes F-measure and minimizes queries.

An optimal strategy for Problem 1 is also optimal for Problem 2, making Problem 1 more general.[3] For instance, strategy S_1 in Example 1 is optimal for both problems, whereas S_2 is optimal only for Problem 2.[4] The theory in [18] yields that both problems require at least $n - k$ questions for growing all k clusters (i.e., the size of a spanning forest of C^+) and at least $\binom{k}{2}$ extra questions for proving that clusters represent different entities. Intuitively, a strategy for Problem 2 tries to ask positive queries before negative ones, whereas a strategy for Problem 1 also does this in a connected fashion, growing larger clusters first. We call `ideal()` the optimal

[1] `https://en.wikipedia.org/wiki/Category:Universities_and_colleges_in_Rome`

[2] In the perfect oracle setting, precision is 1 and F-measure is equal to $\frac{2 \cdot \text{recall}}{\text{recall}+1}$

[3] In practice, some strategy can have great performance in the on-line setting at the cost of slightly worse final recall-queries ratio.

[4] For sake of completeness, we also give an example of sub-optimal strategy for Problem 2 and Example 1. Consider S_3 as (r_a, r_d), (r_b, r_d), (r_a, r_f), (r_d, r_f), (r_d, r_e), (r_a, r_b), (r_a, r_c). The second query (r_b, r_d) is somewhat "wasted" in the perfect oracle setting as the corresponding negative edge would have been inferred after (r_a, r_b) by leveraging transitivity.

strategy for the on-line problem. Consider Example 1, `ideal()` first grows the largest cluster $c_1 = \{r_a, r_b, r_c\}$ by asking adjacent edges belonging to a spanning tree of c_1 (that is, every asked edge shares one of its endpoints with previously asked edges). After c_1 is grown, `ideal()` grows $c_2 = \{r_d, r_e\}$ in a similar fashion. Finally, `ideal()` asks negative edges in any order, until all the labels are known, and also $c_3 = \{r_f\}$ can be identified[5].

3.2 Oracle model and query selection

A strategy S can be thought of as a *sequence*[6] of queries. While in the perfect oracle setting S can leverage transitivity, in real CrowdOracle applications, S can only trade-off queries for F-measure. Let T be the set of positive answers collected by S at a given point of the ER process. We can think of two extreme behaviors.

- S skips all the queries that can be inferred by transitivity, that is, T is a *spanning forest* of V. This is necessary and sufficient to resolve the clusters in the absence of answer error, as shown in Example 1.
- S asks exhaustively all the queries in $V \times V$, that is, T is a noisy collection of cliques, and infers clusters by minimizing disagreements (for instance, via correlation clustering).

In the middle of the spectrum, the work in [7] shows that the error of resolution can be minimized if we strengthen the min-cuts of T with "control queries"[7], exploiting the notion of *expander graphs*. Expander graphs are sparse graphs with strong connectivity properties. We first describe spanning forest approaches for error correction and then we summarize the methods in [7].

Spanning forest As discussed at the end of Section 3.1, the goal of a perfect oracle strategy S is two-fold: promoting positive queries and growing clusters sequentially (only for Problem 1). In order to achieve this goal, the query selection of S can be driven by the *recall gain* of discovering that two specific clusters refer to the same entity. Depending on how S estimates the recall gain of a cluster pair $c_u, c_v \in C'$, we can have *optimistic* and *realistic* query selection. The optimistic approach only considers the maximum inter-cluster matching probability, that is, it estimates the recall gain of c_u and c_v as $\max_{u \in c_u, v \in c_v} p_m(u,v)|c_u| \cdot |c_v|$. Selecting queries optimistically can be computationally efficient, and can give good results if matching probabilities are accurate. However, it can perform badly in presence of non-matching pairs having higher probability than matching pairs [6]. To this end, realistic approach uses a robust estimate, based on the notion of *cluster benefit* $\texttt{cbn}(c_u, c_v) = \sum_{u,v \in c_u \times c_v} p_m(u,v)$. We note that if $|c_u| = |c_v| = 1$ then $\texttt{cbn}(c_u, c_v) = p_m(u,v)$, as in the optimistic approach.[8] Difference between optimistic and realistic is illustrated below.

Example 2 (Optimistic and realistic): Consider clusters grown by S_2 of Example 1 after 4 queries $C' = \{r_a, r_b\}, \{r_c\}, \{r_d, r_e\}, \{r_f\}$. Optimistic estimate of recall gain between non-matching $\{r_a, r_b\}$ and $\{r_f\}$ is $2 \cdot 1 \cdot 0.55 = 1.1$, which is comparable to matching $\{r_a, r_b\}$ and $\{r_c\}$, i.e., $2 \cdot 1 \cdot 0.60 = 1.2$. By switching to realistic, we get $\texttt{cbn}(\{r_a, r_b\}, \{r_f\}) = 0.55 + 0.29 = 0.84$ as opposite to $\texttt{cbn}(\{r_a, r_b\}, \{r_c\}) = 0.60 + 0.54 = 1.14$.

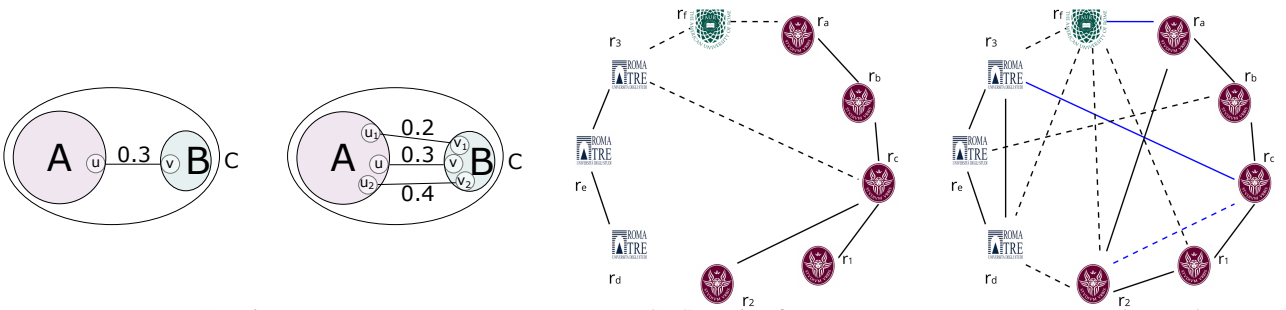
Expander graph Control queries for handling CrowdOracle errors can be selected among those that provide strongest connectivity between records of each cluster, based on the concept of *graph expanders*, which are sparse graphs with formal connectivity properties. Expansion properties of clusters translate into (i) *robustness* since the joint error probability of each cut is small, all the subsets of nodes are likely matching pair-wise;

[5] We note that recall is 1 as soon as c_2 is fully grown.

[6] Partially ordered if parallel.

[7] In addition to the spanning forest.

[8] One may wonder why not take the average. We note that the recall gain is larger for large clusters. However the average would be a robust estimate of the probability that the two clusters are matching.



(a) Joint error (b) Spanning forest (c) Expander graph

Figure 1: (a) Joint error probability. (b-c) Comparison of spanning forest with no error, and expander graphs. Positive (negative) answers are represented with solid (dashed) edges. Wrong answers are shown in blue.

(ii) *cost-effectiveness*: the total number of queries is small, as edge density of expander is small. Technically, different formalizations of connectivity give rise to different notions of expanders. We focus on *edge* expansion.[9]

Definition 2 (Edge expansion): The edge expansion parameter $h(G)$ of a graph $G = (V, E)$ is defined as the minimum cut size $(V', V \setminus V')$ for every subset of nodes $V' \subseteq V$, $|V'| \leq |V|/2$. Cut size is defined as the number of edges crossing the cut. If $h(G)$ is small, we say that the graph has at least one *weak* cut.

G is a γ-expander if $h(G) \geq \gamma$. Intuitively, every subset of the nodes of G that is not "too large" has a "large" *boundary*. In our ER setting, the boundary of a subset of nodes of the same entity provides evidence of how the subset relates to the rest of the graph. A disconnected graph is not an expander (the boundary of a connected component is empty), and every connected graph is an expander. However, different connected graphs have different edge expansion parameters. Consider an entity C being partitioned in two clusters A and B at some point of the ER process, like in Figure 1(a). Upon a positive answer for the pair (u, v) with error probability 0.3, the two clusters are *connected* but the probability of the whole cluster $C = A \cup B$ of being correct is only 0.7. At this point, whatever answers have been collected inside A, the boundary of A consists indeed of a mere edge. Upon two positive answers for the pairs (u_1, v_1), and (u_2, v_2)[10] with error respective probabilities 0.2 and 0.4, the *connection* between A and B becomes more robust. The probability of the whole cluster $C = A \cup B$ can be quantified with the product of the *cut* error probabilities under the assumption of independent error, that is $1 - 0.3 \cdot 0.2 \cdot 0.4 = 0.976$. At this point, the boundary of A consists indeed of three edges. The larger the boundary the higher the success probability. Suppose at this point the ER algorithm decides to merge A and B. Since C is unknown a priori, in order to trust the result, the large boundary property has to hold for *all* the possible partitions A and B. Therefore, the structure we are looking for is an expander. The weights are introduced because the product value, which is related to the weight of the cut, matters more than the cut cardinality (i.e., number of edges). The complete graph has the best expansion property, but it also has largest possible degree, and it would be prohibitively expensive to build. Informally, a graph is a good expander if it has low degree and high expansion parameters. In the following example and in Figure 1(c) we show the possible result of having an expander ($\gamma = 1$) for data in Example 1, compared with spanning forests of Figure 1(b).

Example 3 (Spanning forest and expander graph): Consider the six places of Example 1, plus three extra names (r_1) Studium Urbis, (r_2) Città Universitaria, (r_3) Uniroma 3. Correct clustering is $\{r_a, r_b, r_c, r_1, r_2\}$, $\{r_d, r_e, r_3\}$, and $\{r_f\}$. Both connected components of Figure 1(b) (i.e., trees in the spanning forest) and expander graphs of Figure 1(c) yield the same, correct, clustering. While connected components only work in absence of errors, expander produces the correct clustering also in presence of plausible human errors such as (r_a, r_f)[11],

[9] Other expansion notions include *node* expanders, and *spectral* expanders. We refer the interested reader to [1] for more discussion.

[10] $u_1, u_2 \in A$, and $v_1, v_2 \in B$

[11] "Università di Roma" means "University of Rome".

(r_c, r_3) (false positives) and (r_c, r_2) (false negative). Even though expansion requires more queries than building a spanning forest, it is far from being exhaustive: in the larger clusters, only 5 queries are asked out of $\binom{5}{2} = 10$.

The method `query_cluster()` in [7] is meant to be called in place of `query_pair()` with the purpose of growing clusters with good expansion properties. Given a query (u, v) selected by a strategy (represented with the two corresponding clusters c_u and c_v), `query_cluster()` provides an intermediate layer between the ER logic and the CrowdOracle. Similarly to `query_pair()`, indeed, `query_cluster()` provides functionalities for deciding when two clusters (or any two sets of nodes) are matching. However, instead of asking the selected query (u, v) as `query_pair`(u, v) would do, `query_cluster`(c_u, c_v, β) selects a bunch of random queries between $c_u = c'(u)$ and $c_v = c'(v)$, and returns a YES answer only if the estimated precision of the cluster $c_u \cup c_v$ is high. The parameter β controls the edge expansion value γ trading-off queries for precision.[12] Smaller values of β correspond to sparser clusters, and therefore to less queries ($\beta = 0$ asks a single positive question, yielding result similar to `query_pair()`). Greater values of β correspond to denser clusters and to higher precision. Formally, expected precision increases exponentially with β. We refer the interested reader to Theorem 3 in [7].

Discussion By analogy with the optimistic and realistic spanning forest approaches, we refer to graph expanders as *pessimistic* approach. We note that a CrowdOracle strategy S can leverage multiple approaches together, as discussed later in Section 4. For instance, S can select a cluster pair to compare by using the realistic cluster benefit but then use `query_cluster()` as a substitute of plain connectivity, and so on. Finally, experiments in [7] show that $\beta = 1$ achieves the best progressive F-measure, and we set this as default value.

3.3 Building block algorithms

We now describe the basic operations of our framework. The operations can be implemented either with the simple `query_pair()` oracle interface, or can be modified to apply random expansion with `query_cluster()`.

- **Insert node** This operation grows already established clusters by adding a new node u. Possible outcomes are success, when u is recognized as part of one of the clusters, or fail, in which case $\{u\}$ is established as a new singleton cluster. Specifically, `insert-node`(u) compares u to cluster c_i, $i = 1, 2, \ldots$ until `query_pair`(u, c_i) (or `query_cluster`$(\{u\}, c_i)$, when using the error-correction layer) returns a positive answer. The main lemma in [16] proves that an insert-node-only strategy in the perfect oracle setting requires at most $n - k + \binom{k}{2}$ queries. We can do better by introducing an "early termination" condition (e.g., at most τ comparisons before establishing a new cluster) at the price of possible loss in recall.

- **Merge clusters** Recall of an insert-node-only algorithm can be smaller than 1 for two reasons: (i) positive-to-negative errors of CrowdOracle, and (ii) positive questions "deferred" for early termination. This operation can boost recall by merging pairs of partially grown clusters that represent the same entity. To this end, `merge-clusters`(c_i, c_j) can rely upon a single intra-cluster query `query_pair`(u, v) with arbitrary $u, v \in c_i \times c_j$, or leverage `query_cluster`(c_i, c_j) for expander-like control queries.

In real CrowdOracle applications, the above methods can make mistakes – even when equipped with the pessimistic random expansion toolkit – by adding a node to the wrong cluster or by putting together clusters referring to different entities. Specifically, false negatives can separate in different clusters nodes referring to the same entity, while false positives can include in the same cluster nodes referring to different entities. This is more likely to happen early in the ER process, when we have collected few CrowdOracle answers. Luckily, mistakes can become evident later on, upon the arrival of new answers. The methods below are useful for identifying and correcting `insert-node()` and `merge-clusters()` mistakes.

[12] Technically, β controls the ratio between the edge expansion parameter and the log of the given cluster size.

STRATEGY	COST MODEL	QUERIES	insert-node()	delete-node()	merge-clusters()
wang()	off-line	spanning forest			optimistic
vesd()	on-line		optimistic		
hybrid()	on-line		realistic		optimistic
mixed()	on-line		realistic		realistic
lazy()	off-line	edge expansion	realistic	pessimistic	realistic phase + pessimistic phase
eager()	on-line		realistic/pessimistic	pessimistic	realistic/pessimistic
adaptive()	on-line	both	realistic/pessimistic	pessimistic	realistic/pessimistic

Table 1: Recent strategies categorization. We note that *spanning forest* strategies do not use delete-node(), and that *edge expansion* strategies can leverage optimistic and realistic approaches in some of their phases.

- **Delete node** This operation removes erroneous nodes from the clusters. Specifically, delete-node(u) triggers query_cluster($u, c'(u)$) and whenever it returns NO, it pulls out the node and sets up a new singleton cluster $\{u\}$. We note that delete-node() can successfully fix both single-node errors due to insert-node() failure and larger merge-clusters() errors if one of the clusters c is sufficiently small. In the latter case, we can indeed repeatedly apply delete-node() to remove the smaller cluster, and then put it back together with insert-node() and merge-clusters() operations.
- **Split cluster** In case of severe merge-clusters() errors, we can try to recover by identifying "weak cuts" (see Definition 2) and splitting low confidence clusters into high confidence sub-graphs. To this end, split-cluster(c) selects the minimum cut (c_u, c_v) of a given cluster c (which corresponds to maximum joint error probability) and tries to expand it with query_cluster(c_u, c_v). If it succeeds, no changes to c need do be done. Otherwise, the cluster is split into the two sides of the cut.

In the next section, we will illustrate how combining the above operations leads to various strategies.

4 CrowdOracle Strategies

We now describe prior CrowdOracle strategies using the framework in Section 3, as summarized in Table 1. Then, we summarize the experimental results of the original papers [6, 7] and provide intuitions and illustrative experiments for a new strategy – mixed() – that can outperform hybrid() in specific application scenarios.

4.1 Strategy description

We consider wang(), vesd() and hybrid() from [6] in the perfect oracle setting, and lazy(), eager() and adaptive() from [7] in the general CrowdOracle model. (We refer the interested reader to original papers for more discussion.) For the perfect oracle – or, equivalently, spanning forest – strategies, we also report the key approximation results in [6] for our two problems 1 and 2, under a realistic *edge noise* model for p_m.

Wang The strategy in [18], which we refer to as wang(), is purely based on *optimistic merge clusters* operations. Every node starts as a singleton cluster. Then, cluster pairs are possibly merged in non-increasing order of matching probability, leveraging transitivity. Consider Example 1, where (r_a, r_d) is the edge with highest p_m value. The first operation is merge-clusters($\{r_a\}, \{r_d\}$) – which is equivalent to query_pair(r_a, r_d) in this setting – yielding a negative outcome. The next edge in non-increasing order of matching probability is (r_d, r_f), thus, the second operation is merge-clusters($\{r_d\}, \{r_f\}$), yielding another negative result. Something different happens upon the third operation, which is merge-clusters($\{r_d\}, \{r_e\}$), because

query_cluster(r_d, r_e) yields a positive answer. The singleton clusters r_a and r_d are merged into a "doubleton" cluster $\{r_d, r_e\}$, and – given that the next edge in the ordering is (r_a, r_e) – the fourth operation is merge-clusters($\{r_a\}, \{r_d, r_e\}$) and so on. We note that, being based on spanning forests, the optimistic merge-clusters() variant used by wang() asks only one of the two inter-cluster edges (r_a, r_d) and (r_a, r_e). We can make the strategy *pessimistic* by replacing the query_pair() step with query_cluster().[13]

The wang() strategy has an approximation factor of $\Omega(n)$ for Problem 1 and $O(\log^2 n)$ for Problem 2.

Vesd The strategy in [16], which we refer to as vesd(), leverages *optimistic insert node* operations. Differently from wang(), vesd() starts with a unique singleton cluster, corresponding to the node with highest expected cluster size. In our running example, this is the cluster $\{r_d\}$.[14] Nodes are possibly inserted in current clusters in non-increasing order of expected cluster size: the next node in the ordering is r_e, thus the first operation is insert-node(r_e). This operation triggers query_pair(r_e, r_d) as the first query (differently from wang()) and yields a positive answer, upon which the initial cluster $\{r_d\}$ is "grown" to $\{r_d, r_e\}$. After processing r_a and r_c, the current clusters are $c_1 = \{r_d, r_e\}$ and $c_2\{r_a, r_c\}$. When we consider r_f (which represents a different entity) we have two candidate clusters for insertion, and four intra-cluster edges connecting r_f to nodes in c_1 and c_2. Since optimistic insert node is driven by the highest matching probability edge among those (i.e., (r_a, r_f)) the first cluster selected for comparison – with no success – is $c = \{r_a, r_c\}$. Similarly to wang(), vesd() requires one query for comparing r_f and c, which can be arbitrarily chosen between query_pair(r_a, r_f) and query_pair(r_c, r_f). Before moving to node r_b, insert-node(r_f) compares r_f to the other clusters (i.e., c_2) until possibly success. Otherwise, a new cluster is created. Analogously to wang(), the strategy can be made pessimistic by replacing query_pair() with query_cluster().

The vesd() strategy has better approximation factor of $\Omega(\sqrt{n})$ than wang() for Problem 1 and same $O(\log^2 n)$ for Problem 2. Without assumptions on matching probabilities, vesd() is shown to be $O(k)$ (which is usually $O(n)$) for the off-line setting[15], while wang() can be arbitrarily bad.

Hybrid The hybrid() strategy in [6] combines wang() and vesd(), and can be modified to apply random expansion similarly. Specifically, it first applies a variant of vesd() where insert-node() is modified with a parametric early termination option.[16] That is, some edges between established clusters may be non-resolved (i.e., non-inferable) at some point. In addition, nodes to add are selected based on their singleton-cluster benefit (i.e., sum of incident matching probabilities) with respect to established clusters, making hybrid() a *realistic insert node* strategy. After this phase, recall can be smaller than 1. To this end, hybrid() applies wang() taking care of "deferred" questions due to early termination. The early termination's parameters can be set such that i) hybrid() becomes a realistic variant of vesd(), by letting insert-node() terminate only in case there are no more clusters to consider (that is, no further merge-clusters() operations are needed); 2) hybrid() works like wang(), by inhibiting insert-node() completely; 3) anything in between.

In the worst case, hybrid() provides an $O(\sqrt{n})$-approximation to the on-line Problem 1. If matching probabilities are such that we can pick two representatives from any cluster A before elements of a smaller cluster B, then (no matter what the matching probability noise is) hybrid() performs like ideal(), because the benefit of a third node in the same cluster is higher than any other node in a different cluster, and so on.

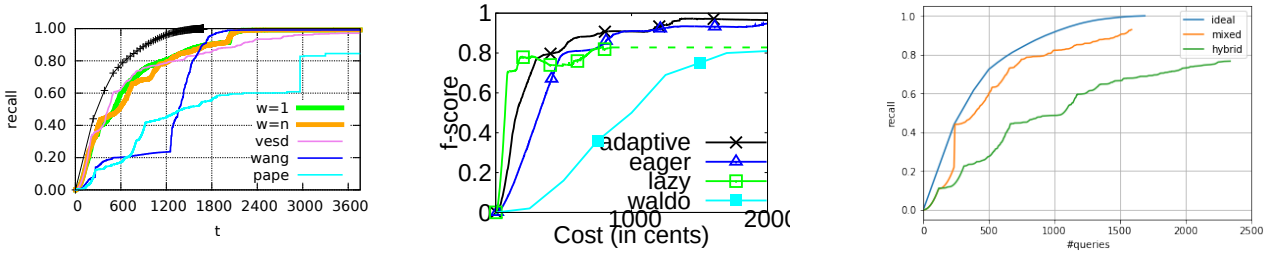
Lazy This pipeline is described in [7] and can be thought of as the simplest edge expansion strategy in the framework. lazy() is indeed focused towards optimizing the progressive F-measure at the cost of lower precision at the start. It does so by following a mix of perfect oracle strategies vesd() and wang() – similarly to what

[13] parameters correspond to the clusters of the edge endpoints

[14] Expected cluster size of r_d is 3.55

[15] In this setting, we only need to minimize the number of queries.

[16] Intuitively, insert-node(u) fails not only if there are no more clusters to examine to, but also if cbn(u, c) drops below a given threshold θ or the number of questions related to node u exceeds a given amount of trials τ.



(a) `hybrid()` (b) `adaptive()` (c) `mixed()`

Figure 2: Experiments on `cora` dataset, featuring $\approx$ 1.9K records and 191 entities. Figure 2(b) considers a subset of 200 records, representing different entities. `pape` and `waldo` strategies are described respectively in [13, 15]

`hybrid()` does – in the beginning to avoid asking extra queries as required to form expanders. However, at the end, `lazy()` runs `merge-clusters()` with `query_cluster()` over all cluster pairs and `delete-node()` over all the nodes, aiming at the correction of recall and precision errors, respectively. We note that the final error correction phase may be challenged by large `merge-clusters()` errors[17], and `delete-node()` could give better results. This was not considered in [7], where `lazy()` is used as a baseline. In the beginning, the only difference with `hybrid()` is `merge-clusters()`, because cluster pairs are possibly merged in non-increasing order of cluster benefit (rather than matching probability), making `lazy()` a *realistic merge cluster* strategy.

Eager The strategy `eager()` in [7] has "orthogonal" behaviour with respect to `lazy()`. Indeed, it maintains high precision at the cost of low progressive F-score. It is a *pessimistic* version of `lazy()` where both `insert-node()` and `merge-clusters()` use `query_cluster()` as a substitute of `query_pair()`. Since expander properties are maintained throughout the execution, large cluster merge errors are unlikely. Therefore, `split-cluster()` is not used and the final error correction phase of `eager()` is the same as `lazy()`.

Adaptive The strategy `adaptive()` in [7] achieves the best of `eager()` and `lazy()` in real CrowdOracle applications. It provides the same final F-measure of `eager()` earlier in the querying procedure, along with the high progressive F-measure of `lazy()`. The intuition is to switch between `query_pair()` and `query_cluster()` depending on the current answer. We compare clusters with `query_pair()` as in `lazy()`, but we use our robust comparison tool `query_cluster()` if the result is in "disagreement" with matching probabilities. Formally, a disagreement can be: (i) a positive answer in case of low average matching probability (< 0.5); (ii) a negative answer in case of high average matching probability (≥ 0.5). `hybrid()` runs two executions of the error correction `merge-clusters()`+`delete-node()` procedure, one at the end (similarly to what `lazy()` and `eager()` do) and another when switching from the insert node phase to the merge cluster phase. Such extra-execution is useful for correcting early errors due to the adaptive nature of the initial `insert-node()` phase.

4.2 Empirical evaluation

Depending on matching and error probabilities (i.e., how accurate machine-based methods and crowd can be on the specific application), the considered strategies may have different performances. `hybrid()` and `adaptive()` are shown to be comparable or better than other strategies in their respective settings. However, when comparable, a user may prefer simpler strategies. Next, we report the main takeaways from [6, 7].

[17] Only removing singleton nodes of one of two erroneously merged clusters, without putting them back together.

1. Suppose there are no erroneous answers, and matching probabilities have uniform edge noise. If the size distribution of clusters is skewed (i.e., there are few large clusters and a long tail of small clusters), we expect `vesd()` to be better than `wang()` and `hybrid()` to be comparable or better than `vesd()`.

2. In the same setting, but with small clusters (for instance, in Clean-Clean ER tasks where most clusters have size 2), we expect `wang()` to be much better than `vesd()` and `hybrid()` to perform like `wang()`.

3. Suppose now the error rate of answers is high and matching probabilities are correlated with the ground truth, that is, truly positive edges have high probability and truly negative edges have low probability. We expect `eager()` to perform better than `lazy()`, and `adaptive()` to perform like `eager()`.

4. Similarly, if the error is high and matching probabilities are uncorrelated with the ground truth, we expect `eager()` to be better than `lazy()`, and `adaptive()` to be like `eager()`.

5. Instead, when the error is low and matching probabilities are correlated with the ground truth, we expect `lazy()` to be better than `eager()`, and `adaptive()` to be like `lazy()`.

6. In the less realistic case where the error is low and matching probabilities are uncorrelated with the ground truth, we still expect `lazy()` to be better, but we expect `adaptive()` to be like `eager()`.

7. There can be mixed cases of reasonable error rate and matching probability noise. We expect the different edge expansion strategies to have similar progressive F-measure in such cases.

Figures 2(a) and 2(b) report an experimental comparison of the considered strategies, against the popular `cora` bibliographic dataset.[18] `cora` is an example of a dataset where matching probabilities are correlated with the ground truth and the cluster size distribution is skewed (the top three clusters account for more then one third of the records), thus matching with application scenarios 1) and 3). The plots confirm the expected behaviour of strategies, with `adaptive()`, `eager()`, `hybrid()`, and `vesd()` being close to `ideal()`.

Mixed Consider the perfect oracle setting, for sake of simplicity. Suppose that the matching probabilities have, in addition to the noise observed in the experiments of Figures 2(a) and 2(b), also a new, systematic noise, which only affects specific clusters and "splits" them in two parts. This can happen in many applications, where real-world entities are represented in well-defined *variations*. Examples include different sweetness – dry, extra dry – of the same wine entity, or the ArXiv and conference versions of the same paper. They are not really different entities, but we can expect lower p_m values between records of the two variations, than between records of the same variation. Systematic "split" error is correlated with entity variations rather than ground truth, and is a challenging scenario for `hybrid()` strategy. After growing the first variation, indeed, if inter-variation p_m values are such that corresponding questions are skipped by `insert-node()` and deferred to the `merge-clusters()` phase, `hybrid()` would seed the new variation as a separate cluster and grow it as if it was a different entity, until the start of the second phase. Our framework enables the design of a new strategy, that we call `mixed()`, that at every step selects `insert-node()` or `merge-clusters()` depending on the realistic cluster benefit, rather than having two separated phases. Experimental comparison of `hybrid()` and `mixed()` is shown in Figure 2(c), against a synthetic version of the `cora` dataset. In the synthetic `cora`, we artificially add the systematic error in the largest cluster c and set $p_m(u, v)$ to 0.001, $u, v \in c$, if u is odd and v is even.[19] The `mixed()` strategy consists of a sequence of *realistic merge clusters* operations, with sporadic *realistic insert node* (with the same early termination as `hybrid()`). Specifically, whenever the next pair of clusters in non-increasing order of benefit (see Section 2) corresponds to two singleton nodes $\{u\}$ and

[18] We refer the reader to the original papers [6, 7] for more details about the dataset and the experimental methodology.

[19] We also augment inter-variation p_m values and re-scale all the other scores in the graph, so that ranking of nodes by expected cluster size does not change for the purpose of the experiment.

$\{v\}$ never seen before, `mixed()` substitutes `merge-clusters`$(\{u\}, \{v\})$ with `insert-node`(w), where w is the largest unprocessed node in non-increasing order of expected cluster size. We observed that `mixed()` performs like `hybrid()` with the original `cora` dataset. However, in the presence of systematic error, after growing the first entity variation c_a, as soon as the second variation of c – which we refer to as c_b – grows large enough that the `cbn`(c_a, c_b) becomes relevant, the two sub-clusters are merged early by `merge-clusters()` into c.

5 Related Work

Entity Resolution (ER) has a long history (see, e.g., [4, 8] for surveys), from the seminal paper by Fellegi and Sunter in 1969 [5], which proposed the use of a learning-based approach, to rule-based and distance-based approaches (see, e.g., [4]), to the recently proposed hybrid human-machine approaches (see, e.g., [17, 9]). We focus on the latter line of work, which we refer to as *crowdsourced ER*, where we typically have access to machine-generated probabilities that two records represent the same real-world entity, and can ask "are u and v matching?" questions to humans.

The strategies described in [16, 18, 6] abstract the crowd as a whole by an *oracle*, which can provide a correct YES/NO answer for a given pair of items. Traditional ER strategies consider ER to be an *offline* task that needs to be completed before results can be used, which can be extremely expensive in resolving billions of records. To address this concern, recent strategies [21, 13] propose to identify more duplicate records early in the resolution process. Such *online* strategies are empirically shown to enable higher recall (i.e., more complete results) in the event of early termination or if there is limited resolution time available. The strategies focus on different ER logics and their performances, which can be formally compared in terms of the number of questions asked for 100% recall [6, 12]. Unfortunately, the strategies do not apply to low quality of answers: if an answer involving two clusters C_1, and C_2, is wrong, the error propagates to all the pairs in $C_1 \times C_2$.

The oracle errors issue raised by [18, 16, 6] is addressed by recent works such as [10, 14, 15, 7]. The solution provided by these works consists of a brand new set of techniques for replicating the same question (i.e. about the same pair) and submitting the replicas to *multiple* humans, until enough evidence is collected for labeling the pair as matching or non-matching (see Section 5). New techniques include voting mechanisms [10], robust clustering methods [14], and query-saving strategies such as classifying questions into easy and difficult [15]. These algorithms show how to make effective use of machine-generated probabilities for generating replicas and correcting errors, sometimes for the specific pair-wise answers and sometimes for the entities as a whole. In this setting, each YES/NO answer for a given pair of items can be interpreted as a matching probability: $1 - p^E$ if the pair is supposed to be matching, and p^E otherwise. (An information theoretic perspective of it is provided in [11].) General purpose answer-quality mechanisms are described in the crowd-sourcing literature [2, 19].

6 Conclusions

In this paper, we considered the pipelines described in [18, 16, 6, 7] in the general CrowdOracle model. We summarized their goals, provable guarantees and application scenarios. Specifically, we described a common framework consisting of simple operations that combined together lead to the considered strategies. This framework raised the issue of a specific scenario, which can be challenging for strategies in [18, 16, 6, 7]. To this end, we leveraged the simple operations introduced in this paper for defining an original strategy, which is better than `hybrid()` in the challenging scenario (and comparable in the traditional setting).

References

[1] N. Alon and J. H. Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

[2] J. Bragg and D. S. Weld. Optimal testing for crowd workers. In *AAMAS*, 2016.

[3] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD Conference*, pages 969–984, 2016.

[4] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[5] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[6] D. Firmani, B. Saha, and D. Srivastava. Online entity resolution using an oracle. *PVLDB*, 9(5):384–395, 2016.

[7] S. Galhotra, D. Firmani, B. Saha, and D. Srivastava. Robust entity resolution using random graphs. In *SIGMOD Conference*, 2018.

[8] L. Getoor and A. Machanavajjhala. Entity resolution: theory, practice & open challenges. *PVLDB*, 5(12):2018–2019, 2012.

[9] 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 Conference*, pages 601–612, 2014.

[10] A. Gruenheid, B. Nushi, T. Kraska, W. Gatterbauer, and D. Kossmann. Fault-tolerant entity resolution with the crowd. *arXiv preprint arXiv:1512.00537*, 2015.

[11] A. Mazumdar and B. Saha. Clustering via crowdsourcing. *CoRR*, abs/1604.01839, 2016.

[12] A. Mazumdar and B. Saha. A theoretical analysis of first heuristics of crowdsourced entity resolution. *AAAI*, 2017.

[13] T. Papenbrock, A. Heise, and F. Naumann. Progressive duplicate detection. *Knowledge and Data Engineering, IEEE Transactions on*, 27(5):1316–1329, 2015.

[14] V. Verroios and H. Garcia-Molina. Entity resolution with crowd errors. In *ICDE Conference*, pages 219–230, April 2015.

[15] V. Verroios, H. Garcia-Molina, and Y. Papakonstantinou. Waldo: An adaptive human interface for crowd entity resolution. In *SIGMOD Conference*, 2017.

[16] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.

[17] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[18] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD Conference*, pages 229–240, 2013.

[19] P. Welinder, S. Branson, S. Belongie, and P. Perona. The multidimensional wisdom of crowds. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems*, NIPS'10, pages 2424–2432, USA, 2010. Curran Associates Inc.

[20] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[21] S. E. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *Knowledge and Data Engineering, IEEE Transactions on*, 25(5):1111–1124, 2013.

[22] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD Conference*, pages 219–232. ACM, 2009.

Human-in-the-loop Rule Learning for Data Integration

Ju Fan
Renmin University of China
fanj@ruc.edu.cn

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

Abstract

Rule-based data integration approaches are widely adopted due to its better interpretability and effective interactive debugging. However, it is very challenging to generate high-quality rules for data integration tasks. Hand-crafted rules from domain experts are usually reliable, but they are not scalable: it is time and effort consuming to handcraft many rules with large coverage over the data. On the other hand, weak-supervision rules automatically generated from machines, such as distant supervision rules, can largely cover the items; however, they may be very noisy that provide many wrong results. To address the problem, we propose a human-in-the-loop rule learning approach with high coverage and high quality. The approach first generates a set of candidate rules, and proposes a machine-based method to learn a confidence for each rule using generative adversarial networks. Then, it devises a game-based crowdsourcing framework to refine the rules, and develops a budget-constraint crowdsourcing algorithm for rule refinement at affordable cost. Finally, it applies the rules to produce high-quality data integration results.

1 Introduction

Despite that machine learning (ML)-based solutions are widely used for data integration tasks due to high effectiveness, the rule-based approach is often preferred as it provides better interpretability that enables interactive debugging. According to a survey [3] over 54 information extraction vendors, 67% of the tools are rule-based while only 17% are ML-based. In addition, Walmart applies rule-based approaches in their product classification to allow domain experts to improve the system [21]. For example, entity extraction aims to extract entity from textual data. The rules like "C `including` e" can be used to extract entities, e.g., `Canon 40D` and `Nikon D80`, for an entity type, say `camera`. In entity matching, it is usually necessary to generate blocking rules to discriminate entity pairs that cannot match.

There are two widely-used methods to construct rules: *hand-crafted* rules from domain experts and *weak-supervision* rules automatically generated by machines. Hand-crafted rules ask experts to write domain-specific labeling heuristics based on their domain knowledge. However, hand-crafted rules are not scalable, especially for large datasets, as it is time and effort consuming to handcraft many rules with large coverage. Therefore, weak-supervision rules automatically generated by machines are introduced [19, 18], e.g., distant supervision rules in information extraction. Weak-supervision rules can largely cover the items; however, they may be very unreliable that provide wrong labels.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

For effective rule generation, we first generate a set of candidate rules using machine algorithms and then use a human-in-the-loop approach to select high-quality rules. There are two important objectives to select *high-quality* rules. 1) *high coverage*: select the rules that cover as many items as possible to label the data. 2) *high quality*: select the rules that induce few wrong labels on their covered items. It is easy to obtain the coverage of the rules but it is hard to get the quality of the rules, as the ground-truth of items covered by the rules is unknown. To address this problem, we propose a human-in-the-loop framework.

We first utilize a machine-based rule evaluation phase to provide a *coarse evaluation* on rule quality (Section 3): how likely a rule can correctly label items. A naïve evaluation method solely based on the training data is far from sufficient, as the training data is usually very limited. To address the problem, we propose to infer labels for the data items not included in the training set, and take inferred labels as the basis for rule quality estimation. We fulfill the label inference by leveraging a recently proposed learning mechanism, generative adversarial networks (GAN) [10, 23]. The idea is to learn a label generative model that captures what *feature characteristics* of items can generate a particular label, using an adversarial game process. Moreover, we also devise a recurrent neural networks (RNN) based encoder-decoder neural network model to realize the GAN mechanism, and the model can alleviate the tedious burden of feature engineering.

However, rule evaluation based on inferred data labels may not be accurate because the inference is computed based on the static training data and may not be effectively adapted to others. Thus, the framework devises an active crowdsourcing approach (Section 4) for rule refinement to further select a subset of "high-quality" rules, by leveraging the human intelligence from crowdsourcing. A straightforward method asks the crowd to answer a *rule validation* task, which asks the crowd to check whether a rule with high coarsely-evaluated precision is valid or invalid. However, the crowd may give low-quality answers for a rule validation task, because a rule may cover many items and the workers cannot examine all the items covered by the rule. To alleviate this problem, we can ask the crowd to answer an *item checking* task, which asks the crowd to give the label of an item and utilizes the result to validate/invalidate rules that cover the item. However it is expensive to ask many item checking tasks. To address this problem, we devise a two-pronged crowdsourcing task scheme that effectively combines these two task types. We also develop a crowdsourcing task selection algorithm to iteratively select rule-validation and item-checking tasks until the crowdsourcing budget is used up.

Finally, in the third phase, we use the selected rules to annotate data with labels (Section 5.1). The challenge here is that an item can be covered by rules with diverse quality or even with conflicting labels. For example, in entity extraction, one rule may provide label 1 while another one labels -1. To address this problem, we develop an *aggregation* approach to consolidate the rules and provide high-quality labels for the items.

2 Overview of Human-in-the-loop Rule Learning for Data Integration

Most of the data integration tasks can be formalized as the problem that assigns a set of data items with one of the two possible labels, -1 and 1. We also call items with label 1 (-1) positive (negative) items. Figure 1 illustrates two data integration tasks, *entity matching* and *entity extraction*. A more detailed entity matching example is shown in Figure 2: It aims to identify any pair of product records (Figure 2(a)) is matched (i.e., being the same product entity) or not. Here, we regard each item as a possible record pair, as shown in Figure 2(b). Moreover, entity extraction task aims at identifying the entities (e.g., `Canon 40D` and `Nikon D80`) that belong to a specific class (e.g., `camera`) from the unstructured text. Formally, given a target class, we can regard this task as a labeling problem that assigns an entity with 1 if the entity is of the class, or -1 otherwise.

Rule-Based Data Integration Approach. The rule-based approach is often preferred as it provides better interpretability that enables interactive debugging. More formally, a rule is defined as a function $r_j : \mathcal{E} \rightarrow \{L, \texttt{nil}\}$ that maps item $e_i \in \mathcal{E}$ into either a label $L \in \mathcal{L}$ or `nil` (which means r_j does not cover e_i). In particular, let $\mathcal{C}(r_j) = \{e | r_j(e) \neq \texttt{nil}\}$ denote the covered item set of r_j, $\mathcal{C}(\mathcal{R}) = \{e | \exists r \in \mathcal{R} | r(e) \neq \texttt{nil}\}$ denote the covered set of a rule set $\mathcal{R}$, and l_i denote the true label of e_i,

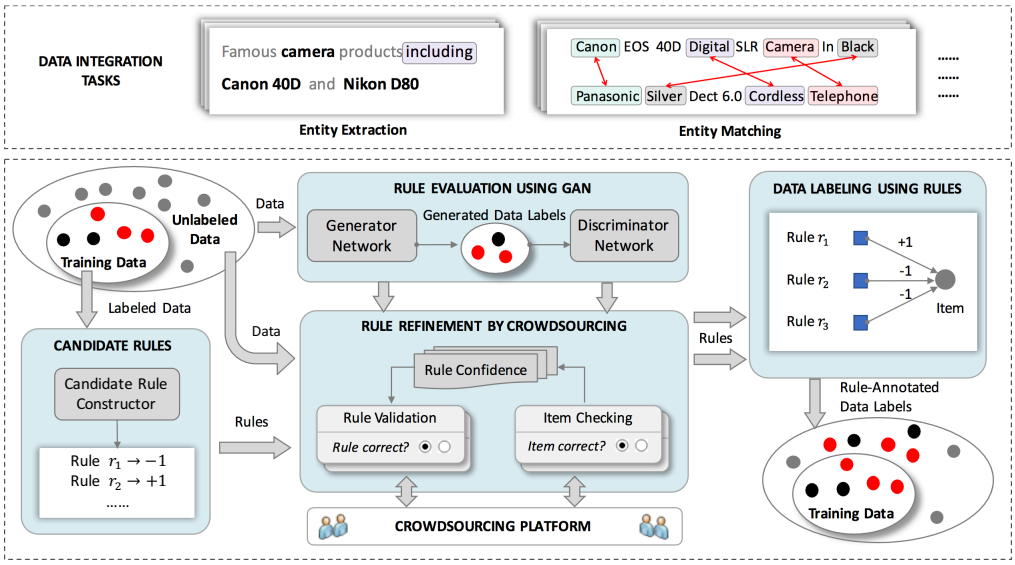


Figure 1: Framework of Human-in-the-loop Rule Learning.

There are two widely-used ways to generate rules for data integration: *hand-crafted* rules from domain experts and *weak-supervision* rules automatically generated by machines. Hand-crafted rules ask users to write domain-specific labeling heuristics based on their domain knowledge. However, hand-crafted rules are not scalable, especially for large datasets, as it is time and effort consuming to handcraft many rules with large coverage. *Weak-supervision* rules automatically generated by machines are introduced [19, 18]. Weak-supervision rules can largely cover the items; however, some of them may be very unreliable that provide wrong labels.

Example 1 (Weak-Supervision Rule): For entity extraction, *distant supervision* [17] is commonly used to identify some textual patterns as *extraction rules* based on a small amount of known positive entities (label $+1$) and negative ones (label -1). For example, considering the text "famous camera products including Canon 40D" and a known "`camera`" entity `Canon 40D`, we may obtain a rule that "C `including` e" indicates that e is an entity of class C. For entity matching, *blocking rules* are also extensively used to eliminate the record pairs that could not be matched. For example, a camera from manufacture `Canon` cannot be matched with another one with `Nikon`, and thus we can use this rule to eliminate record pairs across `Canon` and `Nikon`.

Human-in-the-loop Rule Learning. Note that candidate rules, especially weak-supervision rules automatically generated by machines, could be unreliable. For instance, pattern "C `including` e" for entity extraction may not always indicate e is an entity of class C, and thus incur false positives. On the other hand, a bad blocking rule, such as using `color` to discriminate products, may introduce many false negatives.

To formalize rule quality, we introduce *rule confidence*, denoted by λ_j, of a rule r_j as the ratio of the items in $\mathcal{C}(r_j)$ correctly annotated with label L of r_j, i.e., $\lambda_j = \frac{\sum_{e_i \in \mathcal{C}(r_j)} \mathbb{1}_{\{l_i = L\}}}{|\mathcal{C}(r_j)|}$, where $\mathbb{1}_{\{l_i=L\}}$ is an indicator function that returns 1 if $l_i = L$ or 0 otherwise. Rule confidence is rather challenging to evaluate because most of the data items are unlabeled (denoted as gray circles in the figure).

To address this challenge, we introduce a human-in-the-loop framework, as shown in Figure 1. The framework takes a set of data items $E = \{e_1, e_2, \ldots, e_m\}$ as input, where a subset of data items is labeled with ground truth as the *training data*. In particular, we use black (red) circles to represent data items with label 1 (-1), and gray circles as unlabeled items. The framework also considers a set of candidate rules, which are constructed in either hand-crafted or weak supervision ways. To effectively label data items, it utilizes a rule learning approach that consists of the following three phases.

- *Phase I - Rule Evaluation using Generative Adversarial Networks (GAN):* This phase aims to utilize machine-based methods to provide a *coarse evaluation* of rule confidence. A straightforward way is to

ID	Product Name
s_1	Sony VAIO Silver Laptop
s_2	Apple Pro Black Notebook
s_3	Apple MacBook Pro Silver Laptop
s_4	Apple 8GB Silver 4G iPod
s_5	MacBook Pro Notebook Silver

(a) product records.

Item	True Label	Item	True Label
$e_1 = (s_1, s_2)$	-1	$e_6 = (s_2, s_4)$	-1
$e_2 = (s_1, s_3)$	-1	$e_7 = (s_2, s_5)$	1
$e_3 = (s_1, s_4)$	-1	$e_8 = (s_3, s_4)$	-1
$e_4 = (s_1, s_5)$	-1	$e_9 = (s_3, s_5)$	1
$e_5 = (s_2, s_3)$	1	$e_{10} = (s_4, s_5)$	-1

(b) items (record pairs).

Blocking Rule	Coverage
r_1: (Sony, Apple)	e_1, e_2, e_3
r_2: (VAIO, MacBook)	e_2, e_4
r_3: (Black, Silver)	e_1, e_5, e_6, e_7
r_4: (Laptop, Notebook)	e_1, e_4, e_5, e_9

(c) candidate blocking rules.

Figure 2: A running example for rule learning in entity matching task.

directly use the training data: a rule is of high confidence if it provides correct labels for most of the items in the training set, or vice versa. However, in most of the data integration tasks, the training set is usually far from sufficient to provide effective rule evaluation. To address this problem, our framework *infers* labels for the unlabeled items using a recently proposed learning approach, generative adversarial networks (GAN) [10, 23]. Intuitively, the objective is toward that one cannot easily distinguish whether an item is annotated with the true or inferred label. Then, fed with the inferred labels, it provides the coarse evaluation for rule quality. We will present the details of this phase in Section 3.

- *Phase II - Rule Refinement by Crowdsourcing:* Rule evaluation in the previous phase may not be accurate because the inference is computed based on the static training data and may not be effectively adapted to other items. We devise an active crowdsourcing approach for rule refinement that selects a subset of "high-quality" rules from the candidates. A straightforward method asks the crowd to answer a *rule validation* task, which asks the crowd to check whether a rule with high coarsely-evaluated confidence is valid or invalid. However, the crowd may give low-quality answers for a rule validation task, because a rule may cover many items and the workers cannot examine all the items covered by the rule. To alleviate this problem, we can ask the crowd to answer an *item checking* task, which asks the crowd to give the label of an item and utilizes the result to validate/invalidate rules that cover the item. However it is expensive to ask many item checking tasks. To address this problem, we devise a two-pronged crowdsourcing task scheme that effectively combines these two task types. We also develop a crowdsourcing task selection algorithm to iteratively select rule-validation and item-checking tasks until the crowdsourcing budget is used up. We discuss the techniques in this phase in Section 4.

- *Phase III - Data Labeling using Rules:* This phase applies a rule-based labeling model with the learned rule confidence to obtain "high-quality" item labels. Note that some items may not be labeled by rules either because no rules cover the items, or because the labels provided by rules with low confidence. Thus, it calls for an *aggregation* approach to consolidate the rules. We discuss the approach in Section 5.1.

Figure 2(c) illustrates some "blocking rules" for entity matching. Each rule, represented by two keywords, expresses how we discriminate the product pairs covered by the rule. For example, $r_1 : (\texttt{sony}, \texttt{apple})$ expresses the following blocking rule: any item, say $e_1 = \{s_1, s_2\}$, that satisfies s_1 containing `sony` and s_2 containing `apple` (or s_1 containing `apple` and s_2 containing `sony`) cannot be matched.

Example 2 (Human-in-the-loop Rule Learning for Entity Matching): We illustrate how the proposed framework works for the entity matching task in Figure 2. In the first phase, based on a small training set $\{e_5, e_6, e_8\}$ with known labels, the framework utilizes a GAN-based approach to infer labels for the unlabeled items. The intuition is to learn a generative model that describes "how is a matched pair like" from the training set, and then use the model to label the items. In the second phase, the framework can either ask the crowd to directly validate a rule, e.g., whether $(\texttt{laptop}, \texttt{notebook})$ is effective for blocking, or utilize the crowd to check items covered by rules. Given a crowdsourcing budget, the framework unifies these two task types to further refine rule quality evaluation. Finally, the third phase determines whether to use the rules, so as to balance the following

tradeoff: using more rules may improve the coverage, while increase the risk of incurring error labels. Finally, the framework outputs the labeled items as the result.

3 Machine-Based Rule Evaluation

Rule confidence evaluation heavily relies on data labels: a rule is of high confidence if it provides correct labels for most of the items it covers, or vice versa. However, in most data integration tasks, the amount of labeled items in the training data is usually very small, and thus evaluation solely using the training data is far from sufficient. To address this problem, we propose to infer labels for the items not included in the training data, and take this as the basis for rule confidence estimation. We fulfill item label inference by developing a GAN-based approach. The objective is to learn a *label generator* that captures what *feature characteristics* of items can generate a particular label (e.g., label 1). For example, records s_2 and s_3 in Figure 2(a) share representative words, `Apple` and `Pro`, which can be taken as feature characteristics to generate positive labels. To obtain a good generator, our approach introduces *label discriminator*, and lets it play an adversarial game with the label generator. On the one hand, the generator tries to identify the items with a particular label that look like the ones in our training data, so as to deceive the discriminator. On the other hand, the discriminator tries to make a clear distinction on which items are from the training set and which are generated by the generator. Next, we first present the formalization of our GAN-based learning model, and then introduce RNN-based neural networks that fulfill the model.

GAN-based learning mechanism. We formalize the label generator as a probabilistic distribution over set E that generates positive items, denoted by $p_G(e_i|\theta)$, where θ is the model parameters. Moreover, we use $p_{\texttt{true}}(e_i)$ to represent the ground-truth distribution of positive items in $\mathcal{E}$. Among all items, the larger the $p_{\texttt{true}}(e_i)$ ($p_G(e_i|\theta)$) is, the more likely that e_i can be sampled as a true (generated) positive item. Naturally, the objective of model learning is to compute a good θ that approximates generator $p_G(e_i|\theta)$ to the ground-truth $p_{\texttt{true}}(e_i)$: the positive items that our generator generates are almost the same as the true positive items.

We utilize the generative adversarial networks to effectively learn parameters θ. We introduce a discriminator model $D(e_i)$ as a binary classifier that determines whether e_i is a true positive item. More specifically, $D(e_i)$ outputs a probability that e_i is a true positive item instead of a generative one. The generator and discriminator play an adversarial game as follows. On the one hand, the generator samples items from $\mathcal{E}$ according to the current distribution $p_G(e_i|\theta)$, and mixes the sampled items with the true positive ones in the training data. On the other hand, the discriminator tries to identify which ones in the mixed item set are true positive items. Formally, this can be formalized as a minimax game with a value function $V(G, D)$.

$$\min_G \max_D V(G, D) = \mathbb{E}_{e_i \sim p_{\texttt{true}}(e_i)}[\log D(e_i)] + \mathbb{E}_{e_i \sim p_G(e_i|\theta)}[1 - \log D(e_i)], \tag{5}$$

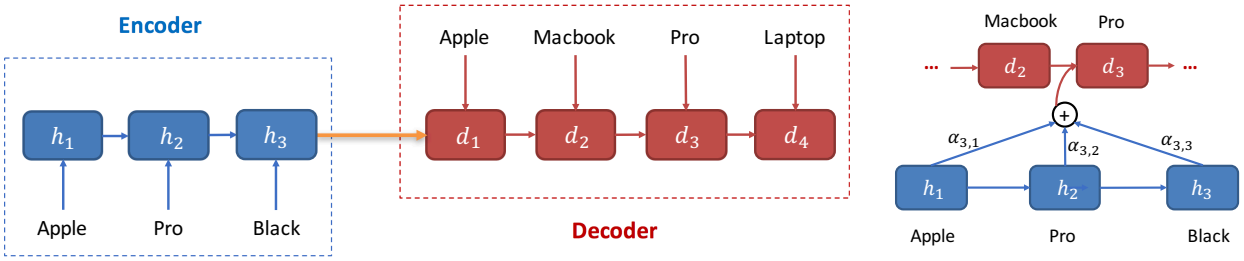
where G and D are the generator and the discriminator model respectively. We defer the discussion on how to derive them later, and focus on presenting how generator and discriminator optimize their objectives now.

Discriminator plays as a *maximizer*: given the current $p_G(e_i)$, it tries to train $D(e_i)$ by maximizing the probability that assigns correct label for both training and generative positive items, i.e.,

$$D^* = \arg\max_D \big(\mathbb{E}_{e_i \sim p_{\texttt{true}}(e_i)}[\log D(e_i)] + \mathbb{E}_{e_i \sim p_G(e_i|\theta)}[1 - \log D(e_i)]\big). \tag{6}$$

Generator plays as a *minimizer*: given the current $D(e_i)$, it tries to deceive the discriminator to believe that a generated e_i is a true positive item. To this end, it trains G by optimizing the following objective

$$\begin{aligned} G^* &= \arg\min_G \big(\mathbb{E}_{e_i \sim p_{\texttt{true}}(e_i)}[\log D(e_i)] + \mathbb{E}_{e_i \sim p_G(e_i|\theta)}[1 - \log D(e_i)]\big) \\ &= \arg\max_\theta \big(\mathbb{E}_{e_i \sim p_G(e_i|\theta)}[\log D(e_i)]\big) \end{aligned} \tag{7}$$



(a) An encoder-to-decoder mode (ENC-DEC). (b) ENC-DEC with attention.

Figure 3: A RNN-based encoder-decoder neural network model for entity matching.

To learn the parameters of generator and discriminator, we can iteratively use the optimization principle, such as stochastic gradient descent (SGD) used in the existing works [10, 23].

RNN-based encoder-decoder model. To apply the GAN mechanism, we need to derive both generative distribution $p_G(e_i|\theta)$ and the discriminator $D(e_i)$. For $p_G(e_i|\theta)$, we can use a method of first computing the likelihood $G(e_i, \theta)$ that e_i is a positive item given model θ, and then applying a softmax function, i.e.,

$$p_G(e_i|\theta) = \frac{\exp(G(e_i, \theta))}{\sum_{e' \in E} \exp(G(e', \theta))} \tag{8}$$

Then, we design neural networks that fulfill both $G(e_i, \theta)$ and $D(e_i)$. We introduce a recurrent neural networks (RNN) based encoder-decoder model [4], as illustrated in Figure 3. For ease of presentation, we use the entity matching task to illustrate how it works. Intuitively, given an item e_i with record pair (s_{i_1}, s_{i_2}), the model computes the probability that s_{i_2} can be "translated" from s_{i_1}. To this end, we model s_{i_1} as a sequence of words, $\mathbf{x} = \{x_1, x_2, \ldots, x_{|s_{i_1}|}\}$, where x_t is an embedding representation (i.e., numeric vector) of the t-th word in the sequence. Similarly, we model s_{i_2} as a sequence of words, $\mathbf{y} = \{y_1, y_2, \ldots, y_{|s_{i_2}|}\}$. For simplicity, we introduce the model for computing $G(e_i, \theta)$, and it can also be applied to compute $D(e_i)$.

We employ a Recurrent Neural Network (RNN) composed by Long Short-Term Memory (LSTM) units to realize the encoder, where each LSTM unit $\mathbf{h}_t$ encodes the *hidden state* of the t-th word in s_{i_1}, i.e., $\mathbf{h}_t = f_{\texttt{LSTM}}(x_t, \mathbf{h}_{t-1})$, where $f_{\texttt{LSTM}}$ denotes the output of the LSTM neural network (we omit its derivation due to the space limit. Please refer to [9] for more details).

The decoder takes the encoded hidden states $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{|s_{i_1}|}\}$ as input, and computes $G(e_i, \theta)$ as the joint probability $G(e_i, \theta) = P(\mathbf{y}|\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{|s_{i_1}|})$ of "decoding" $\mathbf{y}$ from the hidden states. A simple way is to also employ an RNN composed by LSTM units (Figure 3(a)), where each unit is the hidden state of decoding word y_t in s_{i_2}. Then, we compute the joint probability by decomposing it into a sequence of conditional probabilities

$$P(\mathbf{y}|\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{|s_{i_1}|}) = \prod_{t=1}^{|s_{i_2}|} P(y_t|\mathbf{y}_{<t}), \tag{9}$$

where $\mathbf{y}_{<t}$ is the "context" words before y_t, i.e., $\mathbf{y}_{<t} = \{y_1, \ldots y_{t-1}\}$. To obtain $P(y_t|\mathbf{y}_{<t})$, we let $\mathbf{d}_0 = \mathbf{h}_{|s_{i_1}|}$, and then compute

$$P(y_t|\mathbf{y}_{<t}) = \texttt{Softmax}(\mathbf{d}_t); \quad \mathbf{d}_t = f_{\texttt{LSTM}}(y_t, \mathbf{d}_{t-1}) \tag{10}$$

where $\texttt{Softmax}$ is the softmax neural-network layer that maps hidden state $\mathbf{d}_t$ to a probabilistic distribution on the word vocabulary. We may also extend the decoding process with *attention* mechanism, as illustrated in Figure 3(b). The idea is to consider different importance of the encoded hidden states $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{|s_{i_1}|}\}$ when decoding word y_t by introducing a weight $\alpha_{t,r}$, i.e.,

$$\tilde{\mathbf{d}}_t = \sum_{r=1}^{|s_{i_1}|} \mathbf{h}_r \cdot \alpha_{t,r} = \sum_{r=1}^{|s_{i_1}|} \mathbf{h}_r \frac{\exp(\mathbf{d}_t \cdot \mathbf{h}_r)}{\sum_{r'} \exp(\mathbf{d}_t \cdot \mathbf{h}_{r'})} \tag{11}$$

Based on this, the decoder computes the conditional probability as $P(y_t|\mathbf{y}_{<t}) = \texttt{Softmax}([\mathbf{d}_t; \tilde{\mathbf{d}}_t])$, where $[\mathbf{d}_t; \tilde{\mathbf{d}}_t]$ is a concatenation of hidden states $\mathbf{d}_t$ and $\tilde{\mathbf{d}}_t$.

Example 3 (Machine-based rule evaluation using GAN and ENC-DEC): Figure 3 provides examples of the encoder-decoder model. As shown in Figure 3(a), the model uses LSTM units to encode "Apple Pro Black" into hidden states $\mathbf{h}_1$ to $\mathbf{h}_3$, and then uses another set of LSTM units to compute the probability of decoding "Apple Macbook Pro Laptop" from the hidden states. Figure 3(b) illustrates the attention-based model. For instance, for decoding word `Pro`, it considers weights $\alpha_{3,1}$ to $\alpha_{3,3}$. Intuitively, weight $\alpha_{3,3}$ would be smaller than $\alpha_{3,2}$, as `Pro` is not very relevant to word `Black`. We use this encoder-decoder model for computing $G(e_i, \theta)$ and $D(e_i)$, and then apply the GAN mechanism on the training set (e.g., the items with blue color in Figure 2(b)). Intuitively, given a true positive item $e_5 = (s_2, s_3)$, we may use GAN to generate another positive item $e_7 = (s_2, s_5)$. Finally, based on the generated labels, we compute rule confidence for machine-based rule evaluation.

4 Rule Refinement by Game-Based Crowdsourcing

The generative model $G(e_i, \theta)$ learned in the previous section can be used as an estimate of the probability that e_i is a positive item. Then, given a rule r_j with label L, we can estimate its confidence, denoted by $\hat{\lambda}_j$ as

$$\hat{\lambda}_j = \begin{cases} \frac{\sum_{e_i \in \mathcal{C}(\mathcal{R})} G(e_i,\theta)}{|\mathcal{C}(\mathcal{R})|}, & L = 1 \\ 1 - \frac{\sum_{e_i \in \mathcal{C}(\mathcal{R})} G(e_i,\theta)}{|\mathcal{C}(\mathcal{R})|}, & L = -1 \end{cases} \tag{12}$$

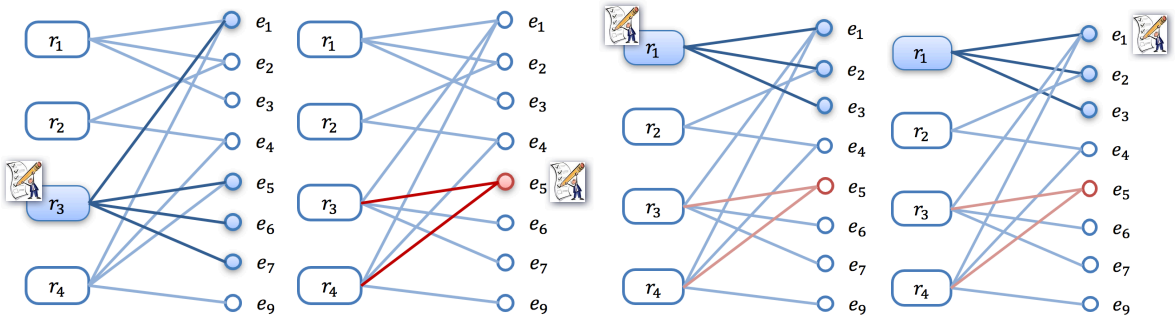
Based on Equation (12), a simple way for rule refinement is to select rules with larger confidence estimates. However, despite of using GAN and RNN, the estimation may not be accurate enough, because the training set is static and thus the inferred labels may not be adapted to other items. Thus, this section presents our active crowdsourcing approach named CROWDGAME for more effective rule refinement.

4.1 Overview of CrowdGame

The objective of CROWDGAME is to leverage the human intelligence in crowdsourcing for selecting high-confidence rules. A naïve crowdsourcing approach is to refine the estimation in Equation (12) by sampling some items covered by a rule, checking them through crowdsourcing, and updating the estimation. However, as the crowdsourcing budget, i.e., the affordable number of tasks, is usually much smaller than item set size, such "aimless" item checking without focusing on specific rules may not be effective.

Two-pronged task scheme. To address the problem, we devise a two-pronged crowdsourcing task scheme that first leverages the crowd to directly validate a rule and then applies item checking tasks on validated rule. To this end, we introduce another type of task, *rule validation*. For example, a rule validation task asks the crowd to validate whether rule r_1 (`sony`, `apple`) in Figure 2(c) is good at discriminating products. Intuitively, human is good at understanding rules and roughly judges the validity of rules, e.g., r_1 is valid as the brand information (`sony` and `apple`) is useful to discriminate products. However, it turns out that rule validation tasks may produce *false positives* because the crowd may not be comprehensive enough as they usually neglect some negative cases where a rule fails. Thus, the fine-grained item checking tasks are also indispensable.

Crowdsourcing task selection. Due to the fixed amount of crowdsourcing budget, there is usually a tradeoff between the rule-validation tasks and item-checking tasks. On the one hand, assigning more budget on rule validation will lead to fewer item checking tasks, resulting in less accurate evaluation on rules. On the other hand, assigning more budget on item checking, although being more confident on the validated rules, may miss the chance for identifying more good rules.



(a) RULEGEN (1st round). (b) RULEREF (1st round). (c) RULEGEN (2nd round). (d) RULEREF (2nd round).

Figure 4: Illustration of game-based crowdsourcing.

We address the challenge by employing two groups of crowd workers from a crowdsourcing platform: one group answers rule validation tasks to play a role of *rule generator* (RULEGEN), while the other group answers item checking tasks to play a role of *rule refuter* (RULEREF). Then, we unify these two worker groups by letting them play a game. On the one hand, RULEGEN tries to elect some rules $\mathcal{R}$ from the candidates $\mathcal{R}^{\mathsf{C}}$ for crowdsourcing via rule validation tasks, such that the elected rules can not only cover more items but also incur less errors. It is easy to examine the coverage of a rule. For the second factor, we take the machine-based rule estimation in Equation (12) to evaluate whether a rule may incur less errors. On the other hand, RULEREF tries to refute its opponent RULEGEN by checking some items that provide enough evidence to "reject" more rules in $\mathcal{R}$. Note that, although also using a game-based strategy, the above process is essentially different from GAN, as it focuses on task selection instead of model (parameters) learning.

Example 4 (Crowdsourced Rule Refinement): Figure 4 shows how CROWDGAME works under a crowdsourcing budget with 4 affordable tasks, which is like a round-based board game between two players. For simplicity, we apply an extreme refuting strategy that one counter-example is enough to refute all rules covering the item, and we consider all rules have the same estimated confidence obtained from Equation (12). In the first round, RULEGEN selects r_3 for rule validation, as it has a large item coverage. However, its opponent RULEREF finds a "counter-example" e_5 using an item checking task. Based on this, RULEREF refutes both r_3 and r_4 and rejects their covered items to maximize the loss. Next in the second round, RULEGEN selects another crowd-validated rule r_1, while RULEREF crowdsources e_1, knows e_1 is correctly labeled, and finds no "evidence" to refute r_1. As the budget is used up, we find $\mathcal{R}_2 = \{r_1\}$ as the refined rule set.

4.2 Crowdsourcing Task Selection

Formalization of rule set loss. Intuitively, We want to select rules with two objectives. 1) high coverage: we would like to select the rules that cover as many items as possible. 2) high confidence: we also prefer the rules that induce few wrong labels on their covered items. For simplicity, we focus on the case that there is only one kind of label L provided by the rules, e.g., $L = -1$ for blocking rules in entity matching.

We formalize the objectives by defining the *loss* of a rule set $\Phi(\mathcal{R})$ as the following combination of the number of uncovered items $|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|$ and the expected number of errors $\sum_{e_i \in \mathcal{C}(\mathcal{R})} P(l_i \neq L)$: $\Phi(\mathcal{R}) = \gamma(|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|) + (1-\gamma) \sum_{e_i \in \mathcal{C}(\mathcal{R})} P(l_i \neq L)$, where γ is a parameter between $[0, 1]$ to balance the preference among coverage and quality. For example, let us consider two rule sets $\mathcal{R}_1 = \{r_1\}$ and $\mathcal{R}_2 = \{r_1, r_3\}$. $\mathcal{R}_1$ covers three items without any errors, while $\mathcal{R}_2$ covers more (6 items) with wrongly labeled items (e_5 and e_7). Note that entity matching prefers quality over coverage on the blocking rules, and thus one needs to set a small parameter γ, say $\gamma = 0.1$ to achieve this preference. Obviously, under this setting, we have $\Phi(\mathcal{R}_1) < \Phi(\mathcal{R}_2)$.

It is non-trivial to estimate $P(l_i \neq L)$. To solve the problem, we use rule confidence to quantify the probability $P(l_i \neq L)$. Moreover, we extend the notation $\hat{\lambda}_j$ to $\hat{\lambda}_j(\mathcal{E}_{\mathsf{q}})$, which denotes an estimator of λ_j for rule r_j based on a set $\mathcal{E}_{\mathsf{q}}$ of items checked by the crowd, and $\hat{\Lambda}^{\mathcal{R}}(\mathcal{E}_{\mathsf{q}}) = \{\hat{\lambda}_j(\mathcal{E}_{\mathsf{q}})\}$ is the set of estimators,

each of which is used for evaluating an individual rule $r_j \in \mathcal{R}$. We will discuss how to estimate $\hat{\lambda}_j(\mathcal{E}_\mathtt{q})$ later. Let $\mathcal{R}^i \subseteq \mathcal{R}$ denote the set of rules in $\mathcal{R}$ covering item e_i, i.e., $\mathcal{R}^i = \{r_j \in \mathcal{R} | r_j(e_i) \neq \mathtt{nil}\}$. For ease of presentation, we denote $P(l_i = L)$ as $P(a_i)$ if the context is clear. Then, we introduce $\Phi(\mathcal{R}|\mathcal{E}_\mathtt{q})$ to denote the *estimated* loss based on a set $\mathcal{E}_\mathtt{q}$ of items checked by the crowd, i.e.,

$$\begin{aligned}\Phi(\mathcal{R}|\mathcal{E}_\mathtt{q}) &= \gamma(|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|) + (1-\gamma)\sum_{e_i \in \mathcal{C}(\mathcal{R})} 1 - P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q})) \\ &= \gamma|\mathcal{E}| - (1-\gamma)\sum_{e_i \in \mathcal{C}(\mathcal{R})} \{P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q})) - \frac{1-2\gamma}{1-\gamma}\} \end{aligned} \tag{13}$$

The key in Equation 13) is $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q}))$, which captures our *confidence* about whether $l_i = L$ (e_i is correctly labeled) given the observations that e_i is covered by rule $\mathcal{R}_i$ with confidence $\Lambda^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q})$. We discuss how to compute $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q}))$ in our technical report [24].

Minimax optimization task selection. Now, we are ready to formalize a minimax objective in the aforementioned RULEGEN-RULEREF game. Let $\mathcal{R}_\mathtt{q}$ and $\mathcal{E}_\mathtt{q}$ respectively denote the sets of rules and items, which are selected by RULEGEN and RULEREF, for crowdsourcing. Given a crowdsourcing budget constraint k on number of crowdsourcing tasks, the minimax objective is defined as

$$\mathcal{O}^{\mathcal{R}^*_\mathtt{q},\mathcal{E}^*_\mathtt{q}} = \min_{\mathcal{R}_\mathtt{q}}\max_{\mathcal{E}_\mathtt{q}} \Phi(\mathcal{R}_\mathtt{q}|\mathcal{E}_\mathtt{q}) \iff \max_{\mathcal{R}_\mathtt{q}}\min_{\mathcal{E}_\mathtt{q}} \sum_{e_i \in \mathcal{C}(\mathcal{R}_\mathtt{q})} \{P(a_i|\hat{\Lambda}^{\mathcal{R}_i}(\mathcal{E}_\mathtt{q})) - \frac{1-2\gamma}{1-\gamma}\} \tag{14}$$

such that task numbers $|\mathcal{R}_\mathtt{q}| + |\mathcal{E}_\mathtt{q}| \leq k$.

Based on Equation (14), we can better illustrate the intuition of CROWDGAME. Overall, CROWDGAME aims to find the optimal task sets $\mathcal{R}^*_\mathtt{q}$ and $\mathcal{E}^*_\mathtt{q}$ from the choices with constraint $|\mathcal{R}_\mathtt{q}| + |\mathcal{E}_\mathtt{q}| \leq k$. RULEGEN prefers to validate rules with large coverage and high confidence to minimize the loss. On the contrary, RULEREF aims to check items which are helpful to identify low-confidence rules that cover many items, so as to effectively maximize the loss. These two players iteratively select tasks until all crowdsourcing budget is used up.

Our minimax objective is fulfilled by asking crowdsourcing tasks, as both $\mathcal{R}_\mathtt{q}$ and $\mathcal{E}_\mathtt{q}$ defined in Equation (14) depend on the crowd answers. We develop an *iterative crowdsourcing* algorithm. The algorithms takes as input a candidate rule set $\mathcal{R}^\mathtt{C}$, an item set $\mathcal{E}$, and a crowdsourcing budget k. It produces the generated rule set $\mathcal{R}_\mathtt{q}$ as output. Overall, the algorithm runs in iterations until k tasks have been crowdsourced, where each iteration consists of a RULEGEN step and a RULEREF step. RULEGEN selects rule-validation tasks. We apply a commonly-used *batch crowdsourcing mode*: We select a b-sized rule set $\mathcal{R}^{(t)}_\mathtt{q}$ that achieves the maximization in Equation (14) at the most. We can prove that the problem of selecting such a rule set is NP-hard, and design an approximate algorithm with theoretical bound to solve it [24]. Similarly, RULEREF selects a batch of b item checking tasks $\mathcal{E}^{(t)}_\mathtt{q}$, so as to achieve the minimization.

Rule Confidence Updating Model. To achieve minimax task selection, we need to estimate rule confidence. Initially, we can use the machine-based estimation in Equation (12) as *prior*. Then, we can use item checking crowdsourcing tasks to continuously update the estimation as more items are checked. The key challenge is how to effectively integrate the crowd's evaluation from rule validation and item checking tasks into the prior. To address this problem, we utilize the *Bayesian estimation* technique [2]. The basic idea is that we use the crowd results as "data observation" to adjust the prior, so as to obtain a *posterior* of rule confidence.

Example 5 (Crowdsourcing Task Selection): Consider the example in Figure 4 with $k = 4$ and $b = 1$. In the 1st iteration, RULEGEN and RULEREF respectively select r_3 and e_5 for crowdsourcing. Based on the crowdsourcing answers, the algorithm updates confidence estimates and continues to the 2rd iteration as shown in Figures 4(c) and 4(d). After these two iterations, the budget is used up, and the algorithm returns $\mathcal{R}_\mathtt{q} = \{r_1\}$.

5 Discussions

5.1 Data Labeling using Rules

This section discusses how we annotate an item e_i given a set $\mathcal{R}_i$ of rules with the estimated confidence $\Lambda^{\mathcal{R}^i} = \{\hat{\lambda}_j\}$ that covers e_i. We first present how to extend crowdsourced rule refinement to more general labeling rules, and then discuss some methods for rule aggregation.

Extension of labeling rules. We discuss a more general case that some rules in the candidates $\mathcal{R}^{\mathtt{C}}$ annotate label -1 (called negative rules for simplicity) while others annotate 1 (called positive rules). Consider our entity extraction example that annotates 1 if an entity belongs to a specific class or -1 otherwise. In this case, an item, e.g., an entity-class pair $(\texttt{Canon 40D}, \texttt{camera})$ could be covered by conflicting rules (textual patterns), e.g., a L_2 rule "`camera` *such as* `Canon 40D`" and a L_1 rule "`cameras` *not including* `Canon 40D`". We devise a simple extension by taking the refinement of positive and negative rules *independently*. More specifically, we split the crowdsourcing budget into two parts: one for positive rules and the other for negative ones. Then, we apply the techniques described in Section 4 to separately learn positive/negative rules using crowdsourcing.

Rule label aggregation methods. After the above rule refinement, an item e_i is now covered by conflicting rules, i.e., a set $\mathcal{R}_i^+$ of positive rules and a set $\mathcal{R}_i^-$ of negative rules. We determine label of e_i based on these two rule set, following the two steps below.

1) Low-confidence rule elimination: We prune rules with $\hat{\lambda}_j < \frac{1-2\gamma}{1-\gamma}$ as they are useless as shown in Equation (14). For example, considering a setting that $\gamma = 0.1$, we would not consider rules with $\hat{\lambda}_j \leq 0.89$, and they would increase the overall loss. As a result, if all rules covering item e_i satisfying the pruning criterion, we leave e_i unlabeled, as no candidate rule is capable of labeling the item.

2) Confidence-aware label aggregation: After the elimination step, if $\mathcal{R}_i^+$ or $\mathcal{R}_i^-$ of item e_i is not empty, an label aggregation method should be applied. A simple method is to simply follow the label of the rule with the maximum estimated confidence, i.e., $r^* = \arg_{r_j \in \mathcal{R}_i^+ \cup \mathcal{R}_i^-} \max \{\hat{\lambda}_j\}$. A more sophistication aggregation method is to apply weighted majority voting [6], where rule confidence is taken the weight in voting. We may also apply a discriminative deep learning model (e.g., multi-layer perceptron, MLP) that takes rules in $\mathcal{R}_i^+$ and $\mathcal{R}_i^-$ as input and trains the aggregation weights, like the existing works [19, 18].

5.2 Candidate Rule Generation for Different Data Integration Tasks

Our human-in-the-loop approach can be easily applied to different data integration tasks. The key of the application is to devise task-specific methods for generating candidate rules. Next, we discuss some representative data integration tasks as follows.

Entity matching task. For entity matching, we want to generate candidate *blocking rules* annotating label -1 to record pairs. Although blocking rules are extensively studied (see a survey [5]), most of the approaches are based on structured data, and there is limited work on generating blocking rules from unstructured text. The idea of our approach is to automatically identify *keyword pairs*, which are effective to discriminate record pairs, from raw text. For example, in Figure 2(c), keyword pairs, such as $(\texttt{Canon}, \texttt{Panasonic})$ and $(\texttt{Camera}, \texttt{Telephone})$, tend to be capable of discriminating products, because it is rare that records corresponding to the same electronic product mention more than one manufacture name or product type.

The challenge is how to *automatically* discover these "discriminating" keyword pairs. We observe that such keyword pairs usually have similar *semantics*, e.g., manufacture and product type. Based on this intuition, we utilize word embedding based techniques [15, 16], which are good at capturing semantic similarity among words. We first leverage the *word2vec* toolkit[1] to generate an embedding (i.e., a numerical vector) for each word, where words with similar semantics are also close to each other in the embedding space. Then, we identify keyword

[1] https://code.google.com/p/word2vec/

pairs from each record pair (s_a, s_b) using the Word Mover's Distance (WMD) [14]. WMD optimally aligns words from s_a to s_b, such that the distance that the embedded words of s_a "travel" to the embedded words of s_b is minimized (see [20] for details). Figure 1 illustrates an example of using WMD to align keywords between two records, where the alignment is shown as red arrows. Using the WMD method, we identify keyword pairs from multiple record pairs and remove the ones with frequency smaller than a threshold.

Relation extraction task. Relation extraction aims to discover a target relation of two entities in a sentence or a paragraph, e.g., `spouse` relation between `Kerry Robles` and `Damien` in a sentence "Kerry Robles was living in Mexico City with her husband Damien". We can use keywords around the entities as rules for labeling 1 (entities have the relation) or -1 (entities do not have the relation). For example, keyword `husband` can be good at identifying the `spouse` relation (i.e, labeling 1), while `brother` can be regarded as a rule to label -1. We apply the *distant supervision* [17], which is commonly used in relation extraction, to identify such rules, based on a small amount of known positive entity pairs and negative ones. For example, given a positive entity pair (`Kerry Robles`, `Damien`), we identify words around these entities, e.g., `living`, `Mexico City` and `husband` between them (stop-words like `was` and `with` are removed), as the rules labeling $+1$. Similarly, we can identify rules that label -1 from negative entity pairs.

Schema matching task. Schema matching aims to discover the semantic correspondences between the columns of two tables [7]. For example, consider a table A with address information `building` and `street`, and another table B with columns `bld` and `st`. Schema matching aims to find column correspondences, (A.`building`, B.`bld`) and (A.`street`, B.`st`). In this task, we can generate candidate rules using both schema- and instance-level information. For schema-level, we can learn string transformation rules from some examples of matched column names, such as (`street`, `st`) using techniques in [1]. For instance-level, we can devise different similarity functions, such as Jaccard, edit distance, and etc., over entity sets of any two columns, e.g., if the Jaccard similarity between the sets is larger than a threshold (e.g., 0.8), the two columns are matched.

Data repairing task. There are many data repairing rules, such as editing rules [8], fixing rules [22], Sherlock rules [13], similarity-based rules [11] and knowledge-based rules [12]. However these rules are data dependent, and they usually work well for some datasets. To produce high-quality, high-coverage, and general rules, we can mix these rules together and use our techniques to effectively select high-quality rules.

6 Conclusion and Future Works

In this paper, we present how to learn high-quality rules for data integration. We first discuss how to generate a set of candidate rules, and then propose a GAN-based method to learn a confidence for each rule. Next we devise a game-based crowdsourcing framework to refine the rules. Finally, we discuss how to apply the rules to address data integration problems.

Next we discuss some research challenges in rule learning for data integration. First, there are many machine learning techniques, e.g., deep learning and reinforcement learning, but they are hard to interpret. So a challenge is how to use machine learning techniques to generate explainable rules. Second, most rules are data dependent and it is hard to transfer the rules from one dataset to another. Thus a challenge is to devise a transfer-learning based method that can apply the knowledge of rule learning on one dataset to another dataset. Third, most rules rely on human knowledge, e.g., synonyms and common senses. A challenge is to build a common-sense base and utilize the common senses to help data integration.

References

[1] A. Arasu, S. Chaudhuri, and R. Kaushik. Learning string transformations from examples. *PVLDB*, 2(1):514–525, 2009.

[2] C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[3] L. Chiticariu, Y. Li, and F. R. Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *ACL*, pages 827–832, 2013.

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[6] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.

[7] J. Fan, M. Lu, B. C. Ooi, W. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE*, pages 976–987, 2014.

[8] 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.

[9] F. A. Gers, J. Schmidhuber, and F. A. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

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

[12] S. Hao, N. Tang, G. Li, and J. Li. Cleaning relations using knowledge bases. In *ICDE*, pages 933–944, 2017.

[13] M. Interlandi and N. Tang. Proof positive and negative in data cleaning. In *ICDE*, 2015.

[14] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *ICML*, pages 957–966, 2015.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[16] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[17] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, pages 1003–1011, 2009.

[18] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, 2017.

[19] A. J. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *NIPS*, pages 3567–3575, 2016.

[20] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.

[21] P. Suganthan, C. Sun, K. Gayatri, H. Zhang, F. Yang, N. Rampalli, S. Prasad, E. Arcaute, G. Krishnan, R. Deep, V. Raghavendra, and A. Doan. Why big data industrial systems need rules and what we can do about it. In *SIGMOD*, pages 265–276, 2015.

[22] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, 2014.

[23] J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang. IRGAN: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*, pages 515–524, 2017.

[24] J. Yang, J. Fan, Z. Wei, G. Li, T. Liu, and X. Du. Rule generation using game-based crowdsourcing. In *Technical Report*, 2018. `http://iir.ruc.edu.cn/~fanj/papers/crowdgame-tr.pdf`.

Call for Papers

35th IEEE International Conference on Data Engineering

8-12 April 2019, Macau SAR, China

The annual IEEE International Conference on Data Engineering (ICDE) addresses research issues in designing, building, managing and evaluating advanced data-intensive systems and applications. It is a leading forum for researchers, practitioners, developers, and users to explore cutting-edge ideas and to exchange techniques, tools, and experiences. The 35th ICDE will take place at Macau, China, a beautiful seaside city where the old eastern and western cultures as well as the modern civilization are well integrated.

Topics of Interest

We encourage submissions of high quality original research contributions in the following areas. We also welcome any original contributions that may cross the boundaries among areas or point in other novel directions of interest to the database research community:

- Benchmarking, Performance Modelling, and Tuning
- Data Integration, Metadata Management, and Interoperability
- Data Mining and Knowledge Discovery
- Data Models, Semantics, Query languages
- Data Provenance, cleaning, curation
- Data Science
- Data Stream Systems and Sensor Networks
- Data Visualization and Interactive Data Exploration
- Database Privacy, Security, and Trust
- Distributed, Parallel and P2P Data Management
- Graphs, RDF, Web Data and Social Networks
- Database technology for machine learning
- Modern Hardware and In-Memory Database Systems
- Query Processing, Indexing, and Optimization
- Search and Information extraction
- Strings, Texts, and Keyword Search
- Temporal, Spatial, Mobile and Multimedia
- Uncertain, Probabilistic Databases
- Workflows, Scientific Data Management

Important Dates

For the first time in ICDE, ICDE2019 will have two rounds' submissions. All deadlines in the following are 11:59PM US PDT.

First Round:

Abstract submission due: May 25, 2018
Submission due: June 1, 2018
Notification to authors (Accept/Revise/Reject): August 10, 2018
Revisions due: September 7, 2018
Notification to authors (Accept/Reject): September 28, 2018
Camera-ready copy due: October 19, 2018

Second Round:

Abstract submission due: September 28, 2018
Submission due: October 5, 2018
Notification to authors (Accept/Revise/Reject): December 14th, 2018
Revisions due: January 11th, 2019
Notification to authors (Accept/Reject): February 1, 2019
Camera-ready copy due: February 22, 2019

General Co-Chairs
Christian S. Jensen, Aalborg University
Lionel M. Ni, University of Macau
Tamer Özsu, University of Waterloo

PC Co-Chairs
Wenfei Fan, University of Edinburgh
Xuemin Lin, University of New South Wales
Divesh Srivastava, AT&T Labs Research

For more details: http://conferences.cis.umac.mo/icde2019/

Data Engineering

It's FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name _____

IEEE Member # _____

Mailing Address _____

_____

Country _____

Email _____

Phone _____

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

TCDE Chair

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398