Data Curation at Scale: The Data Tamer System

Michael Stonebraker
MIT
stonebraker@csail.mit.edu

Daniel Bruckner
UC Berkeley
bruckner@cs.berkeley.edu

Ihab F. Ilyas
QCRI
ikaldas@qf.org.qa

George Beskales
QCRI
gbeskales@qf.org.qa

Mitch Cherniack
Brandeis University
mfc@brandeis.edu

Stan Zdonik
Brown University
sbz@cs.brown.edu

Alexander Pagan
MIT
apagan@csail.mit.edu

Shan Xu
Verisk Analytics
sxu@veriskhealth.com

ABSTRACT

Data curation is the act of discovering a data source(s) of interest, cleaning and transforming the new data, semantically integrating it with other local data sources, and deduplicating the resulting composite. There has been much research on the various components of curation (especially data integration and deduplication). However, there has been little work on collecting all of the curation components into an integrated end-to-end system.

In addition, most of the previous work will not scale to the sizes of problems that we are finding in the field. For example, one web aggregator requires the curation of 80,000 URLs and a second biotech company has the problem of curating 8000 spreadsheets. At this scale, data curation cannot be a manual (human) effort, but must entail machine learning approaches with a human assist only when necessary.

This paper describes Data Tamer, an end-to-end curation system we have built at M.I.T. Brandeis, and Qatar Computing Research Institute (QCRI). It expects as input a sequence of data sources to add to a composite being constructed over time. A new source is subjected to machine learning algorithms to perform attribute identification, grouping of attributes into tables, transformation of incoming data and deduplication. When necessary, a human can be asked for guidance. Also, Data Tamer includes a data visualization component so a human can examine a data source at will and specify manual transformations.

We have run Data Tamer on three real world enterprise curation problems, and it has been shown to lower curation cost by about 90%, relative to the currently deployed production software.

6th Biennial Conference on Innovative Data Systems Research (CIDR '13) January 6-9, 2013, Asilomar, California, USA.

1. INTRODUCTION

There has been considerable work on data integration, especially in Extract, Transform and Load (ETL) systems [4, 5], data federators [2, 3], data cleaning [12, 18], schema integration [10, 16] and entity deduplication [9, 11]. However, there are four characteristics, typically absent in current approaches that we believe future system will require. These are:

- **Scalability through automation.** The size of the integration problems we are encountering precludes a human-centric solution. Next generation systems will have to move to automated algorithms with human help only when necessary. In addition, advances in machine learning and the application of statistical techniques can be used to make many of the easier decisions automatically.
- **Data cleaning.** Enterprise data sources are inevitably quite dirty. Attribute data may be incorrect, inaccurate or missing. Again, the scale of future problems requires an automated solution with human help only when necessary.
- **Non-programmer orientation.** Current Extract, Transform and Load (ETL) systems have scripting languages that are appropriate for professional programmers. The scale of next generation problems requires that less skilled employees be able to perform integration tasks.
- **Incremental.** New data sources must be integrated incrementally as they are uncovered. There is never a notion of the integration task being finished.

These four issues should be addressed in a single coherent architecture, which we call a *data curation system*. The purpose of this paper is to describe Data Tamer, a data curation system motivated by these requirements. In Section 2, we begin with a brief description of three example data curation problems, which Data Tamer is designed to solve. Then, Section 3 continues with the semantic model that Data Tamer implements, followed in Section 4 by a description of the main components of the system. Lastly, Section 5 presents

a collection of experiments on real world curation problems. We conclude in Section 6 with a collection of possible future enhancements.

2. EXAMPLE APPLICATIONS

2.1 A Web Aggregator

This aggregator integrates about 80,000 URLs, collecting information on "things to do" and events. Events include lectures, concerts, and live music at bars. "Things to do" means hiking trails, balloon rides, snowmobile rentals, etc. A category hierarchy of concepts is used to organize this space, and all information is placed somewhere in this hierarchy.

The decision to collect the data from a specific URL is made with a combination of manual and automatic processes that are not relevant to this paper. Once decided, an offshore "wrapper foundry" writes code to extract data from the URL. For each entity that it finds at a given URL, the wrapper outputs a collection of key-value pairs, e.g., (`key1-name`, `value-1`), (`key2-name`, `value-2`), ..., (`keyK-name`, `value-K`). Unfortunately, the source data is rarely web tables, but is usually in pull-down menus, text fields and the like. Hence, site wrappers are non-trivial.

This aggregator needs to federate these 80,000 data sources into a semantically cohesive collection of facts. The 80,000 data sources contain approximately 13M local records with about 200K local attribute names. In addition, local information may be inconsistent, overlapping and sometimes incorrect. Hence, this aggregator faces a difficult data curation problem, which they solve using a collection of ad-hoc and human-centric techniques. The purpose of Data Tamer is to do a better job on this sort of problem than the currently deployed solution at substantially lower cost.

2.2 A Biology Application

A major drug company has 8,000 bench biologists and chemists doing lab experiments. Each maintains a "lab notebook", typically as a spreadsheet where he records his data and observations. Most of the scientists are using different techniques and collecting experiment-specific data such as concentration and density. However, some of these 8000 scientists might well be studying the same reaction or be starting with the same molecule. There is great value to integrating these 8000 sources, so scientists can get a better picture of the reactions they are studying.

Unfortunately, there are no standards for attribute names, no standards for measurement units, and not even a standard for the language for text (English, German, etc...)

These 8000 spreadsheets contain about 1M rows, with a total of 100K attribute names. Again, the scale of the problem makes current data integration tools prohibitively expensive. The goal of Data Tamer is to do a better job at lower cost than current software.

2.3 A Health Services Application

Verisk Health does data integration for the claims records for a collection of 300 insurance carriers. They have manually constructed a global schema for these sources, and are looking to replace their manual processes with more automated ones. In addition, their integrated database contains 20M records, and they wish to consolidate claims data by medical provider. In other words, they want to aggregate all of the claims records, group_by provider. In effect, they want to dedup their database, using a subset of the fields. They are currently doing this task with a lot of manual intervention, and looking for a lower cost, more automated solution. The purpose of Data Tamer is better results at lower cost than their current solution.

3. DATA TAMER SEMANTIC MODEL

3.1 Human Roles

Data Tamer assigns roles for the following humans:

- **A Data Tamer administrator (DTA).** This role is analogous to a traditional database administrator. Hence, the DTA is in charge of assigning roles to the other humans and in deciding what actions to take during the curation process. Specifically, the DTA specifies the collection of data sources that Data Tamer must try to curate.
- One or more **Domain Experts (DE).** These are human domain experts that can be called on to answer questions that arise during the curation process. Each DE has one or more areas of expertise and they are organized into an innovation crowd-sourcing organization as will be explained in Section 4.3.

3.2 Sites and Schemas

Data Tamer assumes that the DTA specifies *sites*, indicated by a URL or file name. Each site is assumed to be a collection of records, each containing one or more key-value pairs. An upstream *wrapper* may be required to construct this format from what the site actually stores. At the present time, Data Tamer is not focused on lowering the cost of such wrappers.

Data Tamer assumes each local data source has information about one entity. Hence, if a source is multi-faceted, then two or more wrappers must be used to ensure that each source contains data about only one entity. If every site describes a different entity, then there is no integration problem. Hence, the goal of Data Tamer is to group local sites into *classes*, which describe the same entity. In Version 1, there has been no effort to identify relationships between entities (such as might be present in foreign keys in an RDBMS) or to deal with other integrity constraints. These extensions are left for future research.

For each class of entity, there are three possible levels of information available. These depend on whether curation is proceeding in a *top-down* fashion or a *bottom-up* fashion. In a top-down model the DTA has information about the schema he is trying to achieve. Hence, the goal is partially or completely specified. In a bottom-up model, such global knowledge is missing and the global schema is pieced together from the local data sources, with perhaps hints available from the DTA. The fact that either model may be used for a given class leads to the following three levels of information.

- **Level 3: Complete knowledge.** In this case, the complete global schema for a given class of entities has been specified by the DTA using a top-down methodology. Generally, the DTA also maps each local data source to a specific class. However, if not done, Data Tamer includes algorithms to perform this task automatically. Although this level of knowledge is available in the Verisk application, we have found level 3 to be quite rare in practice.

- **Level 1: No knowledge available.** In this case, nothing is known about the structure of the classes of information, and a bottom-up integration is utilized. This level of detail might be true, for example, about random HTML tables that were scraped from the web. This is the world of systems like Web tables [8]. Although, this is the level of knowledge in the biology application, we believe it is also uncommon in practice.

- **Level 2: Partial information available.** Using either a top-down or bottom-up methodology, there may be partial information available. There may be specific attributes that are known to be present for some class of entities. This is the case for the Web aggregator, as it requires specific attributes to be present for each node in its classification hierarchy. Alternately, there may be templates available. A template is a collection of attributes that are likely to appear together in one of more classes of entities. For example, a template for a US address might be (number, street, city, state, ZIP code). Note that a template is simply a compound type, i.e., a collection of attributes that usually appear together. Templates may be specified by the DTA as a "hint" or they may be identified via machine learning as noted in Section 4.

3.3 Other Information

In addition, in many domains, there are standard *dictionaries*, which should be used by Data Tamer. A dictionary is a list of data values of some data type that can populate an attribute in some data source. For example, there are 50 states in the United States, about 30,000 cities in the USA, etc. A dictionary is used to specify the name and legal values of such attributes. There are as many dictionaries as the DTA wishes to specify.

Dictionaries are generalized to *authoritative tables*. These are auxiliary data tables that are known to have correct information. For example, a list of (city-name, airport-name, airport-code) could be an authoritative table with three columns.

Furthermore, Data Tamer accommodates *synonyms* of the form XXX is_a YYY. For example, "wages" is_a "salary" or "town" is_a "city". In future versions, we may extend this capability into more general ontologies.

3.4 Management Console and Data Tamer Actions

Sites, categories, templates, dictionaries, authoritative tables, and synonyms can be specified through a *DTA management console*, which is a fairly traditional-looking GUI. A portion of this console is dedicated to allowing the DTA to specify *actions* for Data Tamer to take. These actions are:

- **Ingest** a new data source, and store the incoming data in a Postgres database. In the current version of Data Tamer, this database exists at a single node; however, it is straight-forward to partition this database onto multiple nodes and parallelize the algorithms to be described.

- Perform **attribute identification** on data source-i, as discussed in Section 4.1.

- Perform **entity consolidation** on data source-i, as discussed in Section 4.2

At any time during attribute identification or entity consolidation, Data Tamer may request human assistance from a DE, as discussed in Section 4.3. Lastly, as noted in Section 4.4, any human may visualize any data set currently using a Data Tamer specific interface. We may switch to the more sophisticated Data Wrangler visualization system [1], thereby supporting the manual transformations possible in that system. This issue is further discussed in Section 4.4.

At any time, the DTA may request that attribute identification and/or entity consolidation be redone on all sites. Obviously, this becomes a time consuming task as more sites are present in the Data Tamer system. However, better decisions may be available based on the greater amount of information present. Hence, if source-i or source-j are not specified above, then Data Tamer should run the algorithms to be described on all data sources.

Lastly, Data Tamer keeps a history of all operations performed, and the DTA may "snap" the curation process backwards to any past historical point. This is implemented using a no-overwrite update strategy.

3.5 Training Data

In our conversations with people who have enterprise data curation problems, we have seen two very different scenarios for Data Tamer use. The first one is appropriate for cases with minimal or no advance knowledge (i.e., levels 1 and 2 above). In this case, Data Tamer simply commences operation. Initially it is quite dumb and must ask a human for assistance regularly. As it gets smarter, it asks less often. Moreover, with the benefit of added knowledge, it will often makes sense to go back and run attribute identification and entity resolution again on already processed sites, presumably making better decisions with the benefit of added knowledge. As such, training data is simply accumulated over time by the crowd-sourcing component of Data Tamer.

The second scenario deals with applications where more information is known (level 3 above). In this case, we have observed that real-world applications often have training data available. Specifically, they have collections of entities and/or attributes that are "known duplicates". In other words, they have a collection of pairs of local identifiers that are known to match. There is no guarantee that they have

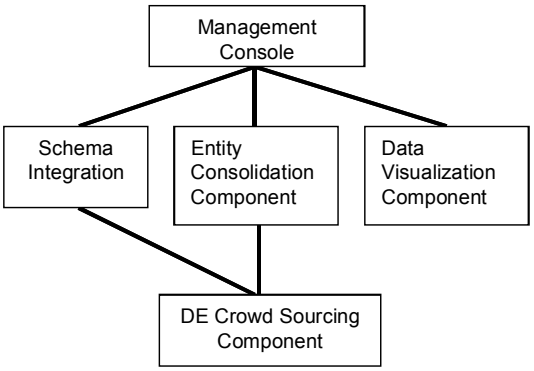

Figure 1: The Data Tamer Architecture

found all of the matches. Hence, they provide a collection of matching attribute names or entities with no false positives. We have observed that the danger of providing a false positive is high in real curation problems, so real world DTAs are very cautious. Therefore, they provide hand-curated collections of known matches.

In the first scenario we start running the Data Tamer system, asking a human for help as appropriate. In the second scenario, we make use of known duplicates as initial training data. We provide more details in Section 4.2.

3.6 Data Source Update

Finally, some data sources may be dynamic, and be continually updated. In this case, Data Tamer can make a new snapshot of a previous data source-k. In this situation, it makes sense to reprocess the data source, since the information has changed. In Version 1. there is no notion of accepting a live data feed. Such an extension is left to a future release.

4. DATA TAMER COMPONENTS

A block diagram of Data Tamer is shown in Figure 1. Indicated in the diagram are the management console and components for schema integration, entity consolidation, DE support and human transformation. These four subsystems are described in this section. Most of the features described in this section are currently operational.

4.1 Schema Integration

The basic inner loop in schema integration is to ingest an attribute, A_i from a data source and compare it to a collection of other attributes in a pairwise fashion. For each pair, we have available both the attribute name and the collection of values. Our approach is to use a collection of algorithms, which we term experts, each returning a score between 0 and 1. Afterwards, the scores are consolidated with a set of weights to produce a composite value. Data Tamer comes with the following four built-in experts, and additional experts may be plugged in via a simple API.

- **Expert-1.** Perform fuzzy string comparisons over attribute names using trigram cosine similarity.
- **Expert-2.** Treats a column of data as a document and tokenizes its values with a standard full text parser. Then, measures TF-IDF cosine similarity between columns.
- **Expert-3.** This expert uses an approach called minimum description length (MDL) that uses a measure similar to Jaccard similarity to compare two attributes [17]. This measure computes the ratio of the size of the intersection of two columns' data to the size of their union. Because it relies on exact matching values between attributes, it is well suited for categorical and finite domain data.
- **Expert-4.** The final expert computes Welch's t-test for a pair of columns that contain numeric values. Given the columns' means and variances, the t-test gives the probability the columns were drawn from the same distribution.

The attributes to be compared depend on which level of information is available to Data Tamer, as described in the next section. Moreover, the DTA can set a threshold for suggested mappings, so that high confidence suggestions can be automatically adopted, while low confidence mappings enter a queue for human review.

4.1.1 Suggesting Attribute Mappings

The attribute mappings to be considered by Data Tamer depend on what information is available for the curation problem at hand, as noted in Section 3.2. Depending on which level is being examined, different tactics are used.

Level 3. In this case, Data Tamer knows the global schema, i.e., all the classes of entities and their associated attributes. Sometimes Data Tamer is told the class to which the incoming data source belongs. In this case, it must merely match the two collections of attributes. The result of running the schema integration component is a pairing of incoming attributes to the elements of the class in the global schema. If Data Tamer is unsure of a pairing, i.e., the matching score is less than a threshold, then a human is involved as noted in Section 4.3.

In other cases, the class to which the incoming entities belong must be identified. In this case, Data Tamer runs the algorithm above on all classes, and computes an aggregate matching score for the attributes. It then picks the best one. Of course, if no score is high enough or if there are two classes with similar scores, then a human is involved in the decision process. It should be noted for each incoming attribute that this algorithm is linear in the number of attributes seen so far. Hence, the total complexity is quadratic. We discuss scalability issues later in this section.

Level 2. In this case, Data Tamer may have certainty on a subset of the attributes. If so, it runs the above algorithms. If an attribute fails to match, it is added to the schema for the class that is specified by the DTA or identified algorithmically. Future data sources can then match a larger collection of attributes. The complexity is the same as for Level 3.

If templates are available, then consider the set, S, of all attributes in any template, plus any dictionary names and synonyms. Data Tamer employs a two-pass algorithm. First, it matches all incoming attributes against the members of S, keeping only the highest scoring one. In a second pass, the score of an incoming attribute is adjusted upward if other attributes match other attributes in the template selected. Then, the match is kept if it is above a threshold.

In addition, Data Tamer watches the incoming sites for collections of attributes that typically appear together. If so, it automatically defines the collection as a new template, and adds the newcomer to the template dictionary.

Level 1. Each incoming attribute is compared against all previously seen attributes, synonyms and dictionaries.

For all levels, the worst case complexity is quadratic in the total number of attributes. The first expert is very cheap to run on pairs of attributes, since it does not look at the data. The other three experts must inspect the data columns and are much more expensive. So far, running time of our attribute identification algorithms has not been a showstopper, since they are run "off line". Our first improvement will be to parallelize them over multiple nodes in a computer network, by replicating the Postgres database and then "sharding" the incoming attributes. This will yield an improvement of a couple of orders of magnitude. After that, further running time improvements will require a two-step process with a cheap first pass and a more expensive second pass on a subset of the attributes.

This two-pass approach will introduce additional experts whose running times are independent of the size of attributes' data sets. The current first expert, which compares attribute names, is an example. Other experts will base comparisons either on attribute metadata or on samples derived from attribute data. Available metadata includes explicit fields in the data set like types and description strings. Although explicit metadata may not be available, useful attribute properties can always be computed and stored in a statistics table. Useful derived metadata include counts and histograms of distinct values, inferred data type, and so on. These statistics are also useful for constructing samples for other experts, for example, an expert that computes the Jaccard similarity of the sets of top-k most common values of two attributes. These first-pass experts will act as a high-pass filter for the more expensive second-pass, and will save wasted effort making detailed comparisons between fields with little in common.

4.2 Entity Consolidation

Entity consolidation is effectively modeled as duplicate elimination. The goal is to find entities that are similar enough to be considered duplicates. This module receives a collection of records, $R_1, \ldots, R_n$, from one or more local data sources that arrive incrementally. We assume that attribute identification has been previously performed. Hence, all records have attributes values from a collection of attributes $A_1, \ldots, A_m$. In general, the data may well be noisy and sparse.

The deduplication process is divided into multiple tasks as we show in the following.

4.2.1 Bootstrapping the Training Process

Initially, the system starts with zero knowledge about the deduplication rules. We learn deduplication rules from a training set of known duplicates and non-duplicates. We assume that duplicate tuples usually have at least one attribute with similar values. We obtain a set of tuple pairs that are potentially duplicates to be presented to expert users as follows. Let Sim_i indicates a similarity measure for values of attribute A_i. For each attribute A_i, we partition the range of Sim_i into a number of equal-width bins, and for each bin we select a sample of tuple pairs with Sim_i belonging to this bin. The obtained pairs, ordered by attribute similarities, are then presented to expert users for labeling. Since the presented pairs are sorted by the attribute similarity in a descending order, the experts could choose to stop labeling pairs below a certain similarity threshold, and declare the remaining unseen pairs as non-duplicates. We denote by T_P the set of pairs labeled as duplicates, and we denote by T_N the set of pairs labeled as non-duplicates.

In order to increase the expected number of found duplicates in the candidate pairs, we only consider the attributes that have relatively large numbers of distinct values when obtaining the candidates (e.g., Title, Address and Phone) while discarding other attributes that are less distinctive (e.g., City and State). The reason is that high similarity on non-distinctive attributes does not increase the chances of being duplicates very much.

Another important source of training data is known duplicates, available with the data sets, as mentioned earlier. In addition, the web aggregator has specified several handcrafted rules that it uses to identify duplicates with high precision. Again, this is a source of known duplicates. We use the existing information as positive training data (i.e., T_P). Negative training data (T_N) is easier to find since non-duplicates are very frequent. Given a random set of tuple pairs, expert users needs only to filter out any non-matching pairs that are highly similar, resulting in negative training data (T_N).

4.2.2 Categorization of Records

Records are classified into multiple categories such that each category represents a set of homogenous entities that have similar non-null attributes and similar attribute values. This might occur, for example, if western ski areas look different than eastern one. For example, vertical drop and base elevation are noticeably different in the two classes of records. In addition, closure because of high winds may be commonly reported for one class and not the other. The benefit of record categorization is twofold: first, by learning deduplication rules that are specific to each category, we achieve higher quality rules that can accurately detect duplicate tuples. Second, we use tuple categorization to reduce the number of tuple pairs that need to be considered in our duplicate detection algorithm. The performance gain is similar to that obtained by blocking techniques (e.g., [7, 14]) currently used in entity resolution in large datasets.

Categorization of records can be done using classifiers. In Data Tamer, categorization is performed in two steps. In

the first step, we obtain a set of representative features that characterize each category. We obtain such features by clustering a sample of the tuples from the available sources. We use a centroid-based algorithm such as k-means++ [6]. The number of categories is determined with the help of the duplicates in training data T_P that was obtained in the bootstrapping stage (Section 4.2.1). In the second step, we assign each tuple to the closest category (w.r.t. to some distance function such as cosine similarity). While similar to blocking in the achieved performance gain, this two-phase categorization is substantially different from previously proposed blocking algorithm, which are usually performed by clustering, indexing, or sorting the entire data set; these are very expensive operations, which we avoid in our categorization algorithm.

Categorization of tuples might change over time when new data sets become available. We maintain the categorization by adding new categories and/or merging/splitting the existing categories when needed. For example, consider the case where the radius of a given category (measured by the maximum distance between the representative features of the category and the category's members) becomes very large compared to the other categories. In this case, we split the category into two or more smaller categories. Efficient incremental categorization is one of our current research directions.

4.2.3 Learning Deduplication Rules

The deduplication rules are divided into two types: (1) cutoff thresholds on attribute similarities, which help pruning a large number of tuple pairs as we show in Section 4.2.4; and (2) probability distributions of attribute similarities for duplicate and non-duplicate tuple pairs. We learn such rules from the collected training data T_P and T_N. For example, one rule indicates that the probability of having two tuples with similar values of 'Title', given that they are duplicates, is close to one. Another rule indicates that having different values for attribute 'State' among duplicates is almost zero. Note that the learning module will choose to neglect a number of attributes that are not useful in learning the probability of being duplicates (e.g., 'user rating' in data collected by the web aggregator). Also, since they are semantically different, the deduplication rules distinguish between missing attribute values and dissimilar attribute values, and we learn the probability of each event separately.

We use a Naïve Bayes classifier [15] to obtain the probability that a tuple pair is a duplicate given the similarities between their attributes. This classifier aggregates the conditional probabilities for all attributes to obtain the marginal probability of being duplicates (assuming conditional independence across attributes).

4.2.4 Similarity Join

The goal of the similarity join between two data sets is retrieving all duplicate tuple pairs. Once the deduplication rules are obtained as shown in Section 4.2.3, we perform similarity join as follows. We obtain all candidate tuple pairs, where each pair belong to the same category and at least one attribute has a similarity above its learned threshold. Then, we compute the attribute similarities of the candidate pairs and we use these similarities to identify duplicate records according to the classifier learned in Section 4.2.3.

Similarity join is performed incrementally to accommodate new data sources that are continuously added. For each new source, we first categorize tuples within the new source, perform self-similarity-join on the new tuples, and perform similarity join between tuples in the new source and tuples in exiting sources. When new training data is added as a result of asking humans for help resolving ambiguous cases, we update the deduplication rules, efficiently identify which tuples are affected by such changes and re-classify them.

4.2.5 Record Clustering and Consolidation

Once we obtain a list of tuple pairs that are believed to be duplicates, we need to obtain a clustering of tuples such that each cluster represents a distinct real-world entity. Clustering of tuples ensures that the final deduplication results are transitive (otherwise, deduplication results would be inconsistent, e.g., declaring (t_1, t_2) and (t_2, t_3) as duplicate pairs while declaring (t_1, t_3) as non-duplicate). We depend on a modified version of a correlation clustering algorithm introduced in [13]. Given a similarity graph whose nodes represent tuples and whose edges connect duplicate pairs of tuples, we perform clustering as follows. The algorithm starts with all singleton clusters, and repeatedly merges randomly-selected clusters that have a "connection strength" above a certain threshold. We quantify the connection strength between two clusters as the number of edges across the two clusters over the total number of possible edges (i.e., the Cartesian product of the two clusters). The algorithm terminates when no more clusters could be merged.

When the underlying similarity graph changes (i.e., new edges are added and/or existing edges are removed), we update the clustering as follows. We identify all nodes in the graph that are incident to any modified edges. Clusters that include any of these nodes are split into singleton clusters. Then, we reapply the same merging operation on the new singleton clusters.

Tuples in each cluster are consolidated using user-defined rules. Null values are first discarded, and then we use standard aggregation methods such as Most-Frequent, Average, Median, and Longest-String to combine the attribute values of tuples within each cluster.

4.3 Human Interface

In both the attribute identification phase and the entity consolidation phase, a human DE may be asked by a DTA to provide curation input. In the case of attribute identification, the task is to decide if two attributes are the same thing or not. In the case of entity resolution, the task is to decide if two entities are duplicates or not. There are two cases that Data Tamer deals with as noted in the next two subsections

4.3.1 Manual Mode

If the tasks requiring human intervention are few or if there are only a few DEs, then the DTA can manually assign human tasks to DEs. He does so by using a collection of rules that specify the classes of issues that should go to specific

individuals. Alternatively, he can just assign tasks manually. In either case, routing of human requests is done by the DTA with minimal infrastructure.

However, if there are many issues, or if DEs vary in the amount of time they have to offer, or if there are many DEs to deal with a large volume of issues, then a more sophisticated crowd sourcing model should be employed. Data Tamer implements the model discussed in the next section.

4.3.2 Crowd Sourcing Mode

Large-scale data curation may require enlisting additional DEs with lesser expertise to help with the workload. Non-guru DEs can be asked to complete "easier" tasks, or they could be crowdsourced to produce results with higher confidence of correctness than could be assumed of any of them individually. But with a large and diverse DE population come the following issues that must be addressed:

- **Determinination of Response Quality.** When a task requiring human intervention is forwarded to a single guru, the resulting response can be assumed to be accurate. But a task addressed by multiple DEs with variable expertise is likely to result in multiple responses of variable quality. Therefore, the set of distinct responses returned by a set of DEs should be accompanied by a probability distribution that reflects the aggregate confidence in each response being correct.
- **Determination of DE Domain Expertise.** The confidence of each distinct response to a task must be a function of the expertise ratings in the given task's domain of those DEs who gave that response. This introduces a challenge for how to characterize and determine domain expertise.
- **Motivation of Helpful and Timely DE Responses.** Given a diverse DE population, individual DEs will differ by the responsiveness and effort put into their responses. The issue here lies in how to incent DEs to be good citizens in responding to tasks.
- **Management of DE Workloads.** DEs not only have variable domain expertise but also variable availability to respond to tasks. Thus, it is necessary to manage workloads in such a way that individual DEs are neither overtaxed nor underutilized given their workload constraints.

We have built a tool (the Data Tamer Exchange, or DTX) which acts as a market-based expert exchange by helping to match a task requiring human input to an individual or crowdsourced set of DEs who can provide it. DTX assumes a collection of attribute identification or entity resolution problems, which must be validated or refuted. For each such task, the tool shows a DTA how many DEs are available in each of some number of fixed Expert Classes associated with the task domain, and how the cost and quality of response will vary according to how many DEs from each class are asked to respond.

The key features of the DTX are the following:

- **Confidence-Based Metrics for DEs and Responses.** The DTX maintains a vector of confidence-based expertise ratings for each DE reflecting their degree of expertise in each of a set of domains specified by a DTA. Each rating is a value between 0 and 1 that denotes the probability that the DE produces a correct response to tasks from the associated domain. A DE's expertise rating for a given domain is calculated from reviews made of each of his responses in that domain by other, more expert DEs and from the DTA who requested the response. A similar confidence-based metric is used to measure the quality of a response (i.e., the probability that the response is correct). A response solicited from a single DE has a quality rating equivalent to the expertise rating of the DE. Crowdsourcing produces a collection of responses (TRUE or FALSE), and the quality rating of each vote is aggregated using Bayesian Evidence Gathering from the expertise ratings of the responders who voted for that choice. More specifically, given a question with n responders with expertise ratings $\vec{E}$ and responses $\vec{R}$, the cumulative confidence of a given response b is:

$$\frac{P(\vec{R}|B) \cdot P(B)}{P(\vec{R}|B) \cdot P(B) + P(\vec{R}|\bar{B}) \cdot P(\bar{B})}$$

such that B and $\bar{B}$ are random variables denoting events whereby response b is a correct and incorrect answer, respectively, and for either random variable (X),

$$P(\vec{R}|X) = \prod_{R_i = X} E_i \cdot \prod_{R_i \neq X} (1 - E_i)$$

such that the subproducts above are the combined probabilities of the correct responders being correct $(R_i = X)$ and the incorrect responders being incorrect $(R_i \neq X)$.

- **Expert Classes.** The DTX dynamically clusters DEs into domain-specific expert classes according to the DE's expertise rating in that domain. For example, the most expert DEs in a given domain might be assigned to expert class #1 on the basis of having an expertise rating of 0.9 or higher. When a task is submitted to DTX, the tool responds by presenting statistics about each expert class in the task domain, including the number of DEs, the cost per DE response (classes with higher expertise ratings charge more per response) and the minimum expertise rating of DEs within the class. On this basis, a DTA decides for each class, how many (if any) DEs he will pay to respond to a task in this class.
- **Economic Incentives for Good Citizenship.** The DTX assumes an economic model whereby the payments earned by a DE for a response is commensurate with his expert class. Payment for responses comes from the accounts of DTAs who are provided a budget with which to complete their tasks. Payment for reviews of DE responses is provided by the system to DEs (at the same rate as responses) and to DTAs (whose pay is added to their budgets).

- **Dynamic Pricing to Manage Workload.** The DTX dynamically adjusts the prices paid per response at each expertise level so as to encourage the selection of underutilized DEs and discourage the selection of overtaxed DEs.

The economic model helps to address two issues with large-scale data curation:

1. Given that responses may be reviewed, DEs are incented to produce helpful and timely responses so as to achieve higher expertise ratings and therefore to earn higher pay.

2. Given that DTAs are assigned a fixed budget with which to complete all of their tasks, they are incented to spend as little as possible for DE responses. This helps to offload the responsibilities of guru DEs by encouraging DTAs to classify tasks according to their difficulty and solicit responses from the least expert (and hence cheapest) DEs whose responses have minimally acceptable confidence.

Further, the payment for reviews incents this crucial input, helping to ensure the accuracy of the confidence-based ratings of DE expertise and response.

4.4 Visualization Component

At any time, the DTA or a DE can call our visualization system and pass a local data source to that system. It displays the data source (or a sample of the source) as a table on the screen. The human can inspect the data source for insight.

It is possible that we will switch the visualization system to Data Wrangler to access their Extract, Transform and Load (ETL) operations. In this way a Data Tamer user could apply Data Wrangler transformations manually and convert data types and formats. In this case, we can remember the transformations applied and put them into a graph of data types. If we see the data types in the future, we can apply the transformations automatically. To allow the DTA to "snap" the data, we always employ a no-overwrite update strategy for data in the Postgres database.

Alternatively, we could implement a Data Tamer-specific visualization interface. In this way, the screen component can be tailored to Data Tamer needs. For example, the entity consolidation system wants to display potentially matching clusters and the schema matcher wants to show columns from multiple local data sources which might match an attribute of interest. Neither capability is included in Data Wrangler.

5. EXPERIMENTAL VALIDATION

We ran the Data Tamer system on the data used by the web aggregator described in Section 2. After modest training on fewer than 50 manually labeled sources, our automatic system successfully identified the correct attribute mapping 90% of the time. This overall success depends on combining results from the individual experts. On their own, attribute name matching is successful 80% of the time, MDL 65%, and fuzzy value matching 55%. Because the aggregator has few numeric columns (postal codes, and latitude/longitude), the T-test expert only provides results for fewer than 6% of attributes, but its results are correct in 65% of cases. The experts compliment each other well: at least one expert identified the correct mapping for 95% of the attributes.

	Aggregator	Data Tamer
Total records	146690	
Pairs reported as duplicates	7668	180445
Common reported pairs	5437	
Total number of true duplicates (estimated)	182453	
Reported true duplicates (estimated)	7444	180445
Precision	97%	100%
Recall	4%	98.9%

Figure 2: Quality results of entity consolidation for the web aggregator data

We evaluated our entity consolidation module using a set of 50 data sources. On average, each data source contains 4000 records, which took 160 seconds to be deduplicated and integrated into the central database (using a single machine). Statistics about the found duplicate pairs in the 50 data source are summarized in Figure 2. We compared our results to the duplicate pairs found by the current deduplication algorithm used by the web aggregator. The total number of records in the 50 data sources is 146690. Data tamer has reported 180445 duplicate pairs, while the aggregator's algorithm only reported 7668 duplicate pairs. The number of common pairs reported by both the aggregator's algorithm and Data Tamer is 5437.

We assume that common pairs are true duplicates. Also, we assume that pairs that were not reported by either algorithm are true non-duplicates. We evaluated the accuracy of the remaining pairs (i.e., pairs reported by one algorithm but not the other) by asking a domain expert to examine a sample of 100 pairs. Based on the expert feedback, 90% of the pairs reported by the web aggregator but not reported by Data Tamer are true duplicates. Also, 100% of the pairs reported by Data Tamer but not by the aggregator were labeled as true duplicates. It follows that the (estimated) number of true duplicates reported by the aggregator is 5437+ (7668-5437)*0.9 = 7444. The number of true duplicates reported by Data Tamer is 5437 + (180445-5437) * 1.0 = 180445. The total number of true duplicates in the data set is 5437+ (180445-5437) * 1.0 + (7668-5437)*0.9 = 182453. The precision of Data Tamer is 180445/180445=100%, while the precision of the aggregator is 7444 / 7668 = 97%. The recall of Data Tamer is 180445/182453 = 98.9%, while the recall of the aggregator is 7444/ 182453= 4%. These results clearly show that our entity consolidation module is capable of significantly increasing the recall of existing deduplication algorithms, while maintaining the same level of precision.

We also ran the schema identification system on the biology problem discussed in Section 2.2. Data Tamer successfully

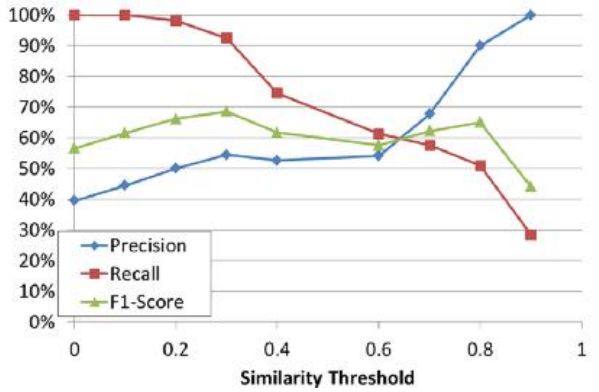

Figure 3: Quality results of entity consolidation for Verisk data

mapped 86% of the attributes.

Lastly, we ran the entity consolidation module on the Verisk medical claims data set. Figure 3 shows the quality of the resulting record pairs for various cut-off thresholds on pairwise similarity. We divided the threshold range [0.1] into 10 equi-width subranges, and we obtain a sample of 15 pairs from each subrange. We relied on a domain expert to classify the sampled pairs into true duplicates and true non-duplicates. We computed the precision, the recall, and the F1-measure for each similarity threshold (Figure 3). To put these results into perspective, we computed the accuracy of the current deduplication algorithm used at Verisk. The precision of this algorithm is 12%, the recall is 30% and the F-score is 17%. On the other hand, our algorithm archives an F-score of 65% at threshold of 0.8.

To gauge user acceptance of our crowd-sourcing exchange, we are executing a two-step evaluation with the biology data mentioned earlier. The company plans to run Data Tamer on their entire biology problem, with several hundred experts as their crowd sourcing component. As a first step, they wanted to do a "dry run" to make sure the system worked correctly. Hence, they asked 33 domain experts to participate in a beta test of the system. We used Data Tamer to perform schema mapping on a portion of their integration problem. To verify Data Tamer's mappings, we used DTX's expert worker allocation algorithms to assign the 33 experts a total of 236 schema-matching tasks. In each task, the user was asked to mark Data Tamer's suggested match as True or False, and if False, to suggest an alternate match. On average, each task was redundantly assigned to 2 experts. This resulted in an average of 7 task assignments per user.

No economic incentives were given to users in this test and participation was voluntary. Of the 33 experts that we contacted, 18 (54%) logged into the system. Each user who logged in performed all of the tasks assigned. In total, 113 of the 236 task assignments were completed and 64% of the tasks received at least one response. The low voluntary response rate suggests the need for economic incentives that reward timely responses. After completion of the assigned tasks, we asked each participant to rate the system's usability on a scale of 1 to 3. The average score given by users was 2.6.

The company is now proceeding with a full scale test of the system using hundreds of domain experts. We plan to cluster data sources by domain, and then leverage each user's data entry history over those sources to determine appropriate initial domain expertise levels. For example, if a user is the creator of a data source in a certain domain, then that user's response can be weighted more heavily in determining the correct answer for a task than responses from users who have not entered data in that domain. We expect to report on this study in the near future.

6. FUTURE ENHANCEMENTS

In the body of the paper we have indicated assorted future enhancements, which we discuss in this section. First, we expect to parallelize all of the Data Tamer algorithms, so they can be run against a sharded and/or replicated Postgres database or against one of the parallel SQL DBMSs. Since many of the algorithms are implemented in SQL or as user-defined functions, this extension is straight forward.

The schema integration module may need to be accelerated by making it a two-step algorithm as noted earlier. In addition, we have yet to consider mechanisms to efficiently redo schema integration. Since our schema integration algorithms are inherently order sensitive, it is entirely possible that a different outcome would be observed with a different ordering of sites. As such an efficient redo will be a required feature.

Our entity consolidation scheme needs to be made incremental, so all of the subtasks can be efficiently run as each new data source is integrated and/or when new training data is added. Moreover, parallelization of this resource intensive module will be especially desirable.

So far, the only data cleaning operations in Data Tamer are in the entity consolidation system. Whenever there are multiple records that correspond to an entity, we can generate a clean result, either automatically or with human assistance. Although this is a valuable service, we need to implement a specific cleaning component. Unfortunately, data cleaning often relies on outlier detection or on a collection of cleaning rules. Outliers are not necessarily errors; for example -99 is often used to indicate that data is missing. Hence, finding an outlier is equivalent in this case to finding a missing value. Also, most cleaning rules are very complex, if they are to be useful. Although it is easy to state that ages and salaries must be non-negative, it is much more difficult to state that temperature near a window should be lower than temperature next to a heat vent, if it is winter. We expect to work on this component in the near future, guided by what users actually find useful in practice.

Similarly, we have not yet systematically addressed data transformations, for example to convert local data to have the same representation, to convert units, or to convert attributes to a common meaning (price without sales tax to total price, for example). Our approach is to maintain a graph of Data Tamer data types. Whenever a user exercises

the visualization system to make a transformation, we plan to remember this as an arc in the graph. Obviously, a user should be able to add arcs to the graph with corresponding code to implement the transformation. This graph could then be used to suggest transformations in the visualization engine.

7. CONCLUSION

This paper has described the main features of Data Tamer, namely a schema integration component, an entity consolidation component, a crowd-sourcing module to organize domain experts, and a visualization component. In the future, we will add more modules to perform data cleaning and reusable transformations.

The system has already been shown to be valuable to three enterprises. At the current time, the code is being adopted by all three companies for production use.

8. REFERENCES

[1] http://vis.stanford.edu/wrangler/.

[2] http://www.compositesw.com/solutions/data-federation/.

[3] http://www.ibm.com/software/data/integration/.

[4] http://www.informatica.com/etl/.

[5] http://www.talend.com.

[6] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA*, pages 1027–1035, 2007.

[7] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. *ACM SIGKDD*, 3:25–27, 2003.

[8] 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.

[9] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *ICDE*, pages 865–876, 2005.

[10] L. Chiticariu, M. A. Hernández, P. G. Kolaitis, and L. Popa. Semi-automatic schema integration in clio. In *VLDB*, pages 1326–1329, 2007.

[11] P. Christen and T. Churches. Febrl. freely extensible biomedical record linkage, http://datamining.anu.edu.au/projects.

[12] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1), 2007.

[13] C. Mathieu, O. Sankur, and W. Schudy. Online correlation clustering. In *STACS*, pages 573–584, 2010.

[14] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[15] T. M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

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

[17] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.

[18] W. E. Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the census, 2006.

APPENDIX

A. DEMO PROPOSAL

At the present time we have validated Data Tamer on three enterprise data curation problems:

- The the web aggregator mentioned in Section 2.1
- The lab notebooks from the "Big Pharma" company mentioned in Section 2.2
- The health claims records mentioned in Section 2.3.

In the demo, we pick the web aggregator application, and we go through the end-to-end steps of data curation, namely:

1. Bootstrap the system by adding a few data sources, and providing initial training data for attribute matching and deduplication modules.
2. Ingest a new data source.
3. Examine the data source visually.
4. Perform schema integration with the composite source assembled so far, asking for human help through our crowdsourcing module.
5. Perform entity consolidation with the composite source assembled so far, again asking for human help in ambiguous cases.
6. We compare our solution with the one currently in production use, and show the superiority of our algorithms.
7. We also illustrate the scaling problem faced by this aggregator, and show why traditional human-directed solutions are problematic.