ORLF: A Flexible Framework for Online Record Linkage and Fusion

El Kindi Rezig
Purdue University
erezig@cs.purdue.edu

Eduard C. Dragut
Temple University
edragut@temple.edu

Mourad Ouzzani, Ahmed K. Elmagarmid
Qatar Computing Research Institute
{mouzzani, aelmagarmid}@qf.org.qa

Walid G. Aref
Purdue University
aref@cs.purdue.edu

Abstract—With the exponential growth of data on the Web comes the opportunity to integrate multiple sources to give more accurate answers to user queries. Upon retrieving records from multiple Web databases, a key task is to merge records that refer to the same real-world entity. We demonstrate ORLF (Online Record Linkage and Fusion), a flexible query-time record linkage and fusion framework. ORLF deduplicates newly arriving query results jointly with previously processed query results. We use an iterative caching solution that leverages query locality to effectively deduplicate newly incoming records with cached records. ORLF aims to deliver timely query answers that are duplicate-free and reflect knowledge collected from previous queries.

I. INTRODUCTION

The sheer amount of information on the Web has triggered the emergence of a vast array of applications that integrate data from multiple Web sources to answer users’ queries. This has brought along new data curation challenges. Data across different Web sources contains multiple anomalies, such as missing data, inconsistent data, and oftentimes the same real world entities, e.g., a company, a person, or a book, is represented differently by different Web sources.

Web applications often require duplicate-free data and error-free representation of entities. The former goal is achieved through Record Linkage (RL) while the latter is achieved through Data Fusion. Record Linkage and Data Fusion (RL&F) are two well-studied problems [2], [4]. While significant effort has been dedicated to solve them, little work has been conducted to perform them at query time. Online RL&F differs from its offline counterpart in the following aspects: (i) the entire data cannot be obtained all at once, e.g., usually only through restricted Web query forms, (ii) since it runs at query time, it has to be performed relatively fast; and (iii) multiple query submissions may be needed to acquire enough evidence about an “ideal” representation of an entity.

We introduced the algorithmic infrastructure of ORLF in [7]. The proposed infrastructure allows seamless application of RL&F over live data coming from multiple Web sources. ORLF leverages previously submitted queries to perform RL&F on query results. As a result, the more queries ORLF processes, the better the query results are likely to be. To achieve this goal, ORLF uses a caching system that serves two main purposes: (i) storing fused records of entities seen so far from previous queries and (ii) allowing a fast lookup at query time to identify relevant entities in the cache and update them with newly arriving query records. We developed a new record indexing data structure to obtain a fast lookup time. The

index, which is based on the B*-tree index [10], is used to index the cache records and to answer top-k similarity queries.

We implemented a software framework for online RL&F that incorporates these algorithmic features and a number of additional ones.

In this demo, we will walk the audience through the following key features of ORLF:

- An end-to-end framework to support RL&F at query time.
- The ability to seamlessly customize our framework by plugging in specific RL and data fusion algorithms. This feature is important because most RL&F algorithms are data-specific.
- A visually rich user interface that allows monitoring the quality of the data in the cache over time.
- We prepared several scenarios from five domains, such as restaurant, hotel and books, to demonstrate that the quality (cached results include previous knowledge about entities) and efficiency (cached results reduce the number of unnecessary RL&F operations) of RL&F greatly benefit from query locality.

II. SYSTEM DESCRIPTION

We show the functioning of ORLF in Figure 1. After obtaining the query results from the Web databases, ORLF first attempts to match them to cached records. If a match is found in the cache for an incoming record $r$ (the Hit path), $r$ is fused jointly with its matching cache record. If a match is not found in the cache for $r$ (the Miss path), ORLF adds $r$ to a miss list $L_i$. Once all records have been consumed, ORLF applies a record linkage and fusion process on $L_i$ and appends the result to the cache. To support these two paths, we introduce several key components.

A. ORLF Components

Caching Policy: We implemented in ORLF three caching policies to support multiple application settings. The caching policies dictate the cache record update behavior of ORLF upon processing new queries: dynamic - all the records in the cache are subject to updates, static - records in the cache are read-only, and static with in-place updates - is similar to the static one except that a subset of the cache records are allowed to be updated until they reach a desired “cleaning factor”.

- dynamic
- static
- static with in-place updates

Record Indexing: We use a B*tree index [10] to index the cache records. The B*tree index enables efficient similarity search.

Query Processing: ORLF processes queries in real-time. The query processing pipeline includes:

1. Query Parsing: The query is parsed to extract relevant information.
2. Record Lookup: ORLF looks up the cache for previously processed records.
3. Fusion: If a match is found, the records are fused. Otherwise, new records are added to the cache.
4. Result Generation: The final result is generated based on the fusion outcome.

Efficiency: ORLF is designed to be efficient in terms of:

- Storage: ORLF minimizes the storage footprint by leveraging previously processed queries.
- Processing: ORLF processes queries in real-time.

Quality: ORLF is designed to maintain high-quality results by leveraging previously processed queries.

Integration: ORLF integrates seamlessly with existing Web applications.

Scalability: ORLF is scalable to handle a large number of queries.
**Record Lookup:** ORLF has two processing paths for the incoming query records: the Hit path and the Miss path (Figure 1). Upon processing an incoming record \( r \), ORLF attempts to find a record that is similar to \( r \) in the cache, so that they can be cleaned collectively. We developed an indexing data structure based on the B\(^d\)-tree index [10] to support fast (approximate) record search (see [7] for details) in the cache. The index is used to retrieve the \( k \) most similar records to \( r \).

**Custom RL:** The set of the top-\( k \) records returned by the index needs to be further filtered out using a matching function. We implemented three domain-specific matching functions (Section III-A). Users however can input their matching algorithm in the form of a Java code to perform Custom RL. The function takes as input two records and returns True if they match (refer to the same entity), and False otherwise.

**Handling Non-Matching Records:** When the top-\( k \) records for \( r \) do not include any matching record, \( r \) is processed in the Miss path. The incoming records that do not have a match in the cache are stored in a data structure \( L_i \), where \( i \) is the source index. RL&F are performed across the records in \( L_k \) where \( 1 \leq k \leq N \), \( N \) is the number of data sources. The resulting fused records are appended to the cache.

**Fusion:** The fusion procedure \( fuse \) is an input to the ORLF system. Users are allowed to feed ORLF their own data fusion algorithms (in the form of a Java procedure). ORLF has a default fusion procedure which uses majority voting as the fusion criterion. The input of \( fuse \) is a collection of records and the output is one fused record.

**ORLF Dashboard:** The dashboard allows users to view and edit the cache configuration. It also provides users with statistics such as the current size of the cache and the average query hit ratio (ratio of records returned by the query that were matched to cache records). The dashboard offers users intuitive visualizations to better understand the state of the cache. For example, Figure 2b shows a heatmap of cached entities over a time window: the darker the entry, the more frequently that entry was matched to query results. This information is useful to assess the effectiveness of the cache in terms of how frequently it matches query results. Figure 2a shows an example visualization of the cache table, the attribute “sources” contains the ratio of sources that contributed in fusing the cache record.

### B. ORLF User Input

**ORLF** accepts four types of user inputs: (1) Schema-related files; (2) Cache warm up; (3) Record matching procedure; and (4) Fusion procedure.

**Schema-Related Files:** ORLF returns an answer to a query \( Q \) over a target schema \( T \). Users input two sets of XML files: the source and target schema and the correspondences between their elements.

**Cache Warm up:** Users may choose to bootstrap the system with a set of queries that are posted to ORLF to populate the cache initially. This input is optional.
Record Matching Procedure: ORLF requires a pair-wise record matching function in order to classify a record pair as a match or a mismatch. Since record matching functions are highly domain-dependent, ORLF allows users to write their own Java code to perform record matching. This function is used to remove false positive record matches from the top-k result set returned by the index data structure for an incoming record. The following is the signature of the function compare:

```java
Boolean compare(Record target_record)
```

Function compare is a non-static member function of the class Record. It returns True if the calling record matches target_record, and False otherwise.

Fusion procedure: ORLF also gives users the flexibility to plug in their own fusion procedure. The fusion operation is performed for each list of matching records. The signature of the function fuse is:

```java
Record fuse(Collection<Record> records)
```

Fuse is a non-static member function of the class Record. It fuses the calling record with a collection of records and returns the fused record. ORLF provides three fusion algorithms: (i) default, based on majority voting, (ii) Solaris’ ACCU fusion algorithm [6], and (iii) LTM and LTMinc [11].

III. DEMONSTRATION OVERVIEW

In this demonstration, the audience will experience two different usage scenarios using three datasets.

A. Datasets

While ORLF works directly with Web databases, we will use three pre-crawled datasets to avoid connectivity issues during the demonstration. These datasets are: (1) Restaurants:¹ We crawled 9 Web databases that provide information about restaurant listings in the metropolitan Chicago area, US, such as Yelp (10,115 records), YellowPages (7,798 records) and Yahoo (10,820 records); (2) Book Author:² This dataset is crawled from abebooks.com. It consists of 1,263 book entities, 2,420 book-author facts, and 48,153 claims from 879 book seller sources; and (3) Flight:² This dataset contains flight data from 38 sources.

B. Demonstration Scenarios

We will showcase two demonstration scenarios: (1) an interactive demonstration in which we walk the audience through the core features of the ORLF system and (2) a scenario where users can set up ORLF for a particular application domain.

1) Scenario I: A Walk-through of the Core Features of ORLF: The audience will interact with ORLF at multiple levels, namely: (1) Record Linkage and Fusion procedures editor; (2) Warming up the cache; (3) Experimenting with different cache configurations, presenting their impact on the data quality of the query results; and (4) Monitoring the state of the cache.

(1) Plug-and-Play Record Linkage and Fusion: We will demonstrate multiple custom RL&F algorithms through the ORLF RL&F rule editor. The audience will select from three fusion algorithms, namely, majority voting (the default fusion), the Solaris’ fusion algorithms [6] and the LTM and LTMinc fusion algorithms [11]. The audience will also have the opportunity to contrast the outcome of the fusion algorithms in ORLF on different datasets.

(2) Warming up the Cache: We will showcase two methods to warm up the cache: (i) import a clean dataset into ORLF and (ii) generate a simulated user query stream to ORLF (we explain this procedure in [7]), post them to the sources, collect the query results, apply off-the-shelf RL&F algorithms, and save the outcome into the cache.

(3) Selecting Cache Configuration: After the warm up phase, the audience will select the cache configuration, and

¹http://cis.temple.edu/~edragut/research.htm
²http://lunadong.com/fusionDataSets.htm
the eviction policy. The goal is to illustrate the outcome of a query result under different cache configurations.

(4) Monitoring the State of the Cache: ORLF features a dashboard that shows the audience key insights about the current state of the cache, these include: (i) the usefulness of the cache entries computed so far, expressed in terms of average hit ratio per query; (ii) freshness information of cache records; and (iii) confidence of fused records based on how many sources were involved in fusing them.

2) Scenario II: Setting up ORLF for a Specific Application Domain: We will walk the audience through the steps required to set up ORLF for a specific application domain. As an example, we will use the Restaurants dataset.

(1) Cache Index: A key part in setting up the B^d-tree-based cache index is the selection of the attributes that are used as a search key in the index. For example, in the restaurant data, we chose the attributes {name, address, phone} to be the search key. After the attribute selection, a few parameters need to be set empirically for the B^d-tree index to answer top-k queries. Details on how to tune these parameters can be found in [7], [10].

(2) Simulated Queries: In order to simulate the influx of queries faced by search engines, we proposed in [7] a method to generate simulated queries. We will give an overview of the procedure. The stream of generated queries will be used to demonstrate the cache warm up.

(3) Cache Warm up: The previously generated queries follow a power-law distribution. The 20% most frequent queries are posted to the Web databases, and their results are cleaned offline using FRIL [5].

(4) Record Matching and Fusion Procedures: As discussed previously, the record matching and fusion procedures need to be set up depending on the application domain. We will show the audience those functions for the restaurant data.

IV. RELATED WORK

In this section we give a brief overview of some related systems built to address record linkage and fusion.

NADEEF/ER [3] is a RL system built as an extension of the open-source generalized data cleaning system NADEEF [3]. It leverages NADEEF’s programming interface to specify entity resolution rules. Users specify such rules based on their knowledge of the data and NADEEF treats them as black boxes to detect duplicates. We adopt the same idea for specifying the entity matching and fusion functions in ORLF. As opposed to ORLF, NADEEF/ER requires all the data to be available at once which makes it unsuitable for query-time RL.

Online entity resolution has been addressed in [1], [9], [8]. In [1], the main goal is to reduce the overhead incurred by resolving record pairs. The authors propose a set of semantics that would avoid resolving some pairs altogether as long as the selected semantics constraints are met.

Authors in [9] propose a query-time cleaning algorithm for aggregate queries. The idea is to use a clean sample of the database to estimate the aggregate attribute value. In [8], record duplicates are identified at query time. The solution employs two classifiers that collaborate with each other iteratively to refine the set of found duplicates.

ORLF differs from the above systems in two key aspects: (i) it performs fusion on top of RL and (ii) it harnesses RL&F results from previous queries.

A set of algorithms to perform online data fusion is proposed in [6]. The key idea is to stop querying further data sources that are unlikely to change the decision about the “correct” fused value computed so far. As opposed to ORLF, these techniques do not perform record linkage and assume the availability of information about the accuracy of the sources and their copying relationships. Because ORLF is orthogonal to the fusion algorithm being used, we have implemented one of these techniques, namely ACCU [6], into our system.

A Bayesian approach to perform data fusion is proposed in [11]. It learns the quality of data sources and incorporates this knowledge in the fusion operation. Because of its stateless nature, the approach falls short in the online setting.

V. CONCLUSIONS

The proposed demonstration brings a new dimension to the tasks of RL and fusion: query time. At its core, ORLF leverages query locality, iterative caching, and smart indexing techniques to efficiently deliver duplicate-free and accurate query results. Moreover, ORLF provides users with the ability to use their own record matching and fusion procedures. To showcase its flexibility, the audience will be able to try the system on multiple application domains and with different record matching and fusion algorithms.

ACKNOWLEDGMENT

This work was supported in part by the U.S. National Science Foundation grants IIS 1117766 and BIGDATA 1546480.

REFERENCES