Discovering Linkage Points over Web Data

Oktie Hassanzadeh
IBM T.J. Watson Research Center
hassanzadeh@us.ibm.com

Ken Q. Pu
UOIT
ken.pu@uoit.ca

Soheil Hassas Yeganeh
University of Toronto
soheil@cs.toronto.edu

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

Lucian Popa
IBM Research – Almaden
lpopa@us.ibm.com

Mauricio A. Hernández
IBM Research – Almaden
mahernan@us.ibm.com

Howard Ho
IBM Research – Almaden
ctho@us.ibm.com

ABSTRACT

A basic step in integration is the identification of *linkage points*, i.e., finding attributes that are shared (or related) between data sources, and that can be used to match records or entities across sources. This is usually performed using a *match* operator, that associates attributes of one database to another. However, the massive growth in the amount and variety of unstructured and semi-structured data on the Web has created new challenges for this task. Such data sources often do not have a fixed pre-defined *schema* and contain large numbers of diverse attributes. Furthermore, the end goal is not schema alignment as these schemas may be too heterogeneous (and dynamic) to meaningfully align. Rather, the goal is to align any overlapping data shared by these sources. We will show that even attributes with different meanings (that would not qualify as schema matches) can sometimes be useful in aligning data. The solution we propose in this paper replaces the basic schema-matching step with a more complex instance-based schema analysis and linkage discovery. We present a framework consisting of a library of efficient lexical analyzers and similarity functions, and a set of search algorithms for effective and efficient identification of linkage points over Web data. We experimentally evaluate the effectiveness of our proposed algorithms in real-world integration scenarios in several domains.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems; H.2.5 [**Database Management**]: Heterogeneous Databases

Keywords

Record Linkage, Entity Resolution, Link Discovery, Schema Matching, Data Integration

1. INTRODUCTION

Many increasingly important data management and mining tasks require integration and reconciliation (or fusion) of data that reside in large and heterogeneous data sources. Data integration is generally defined as combining data to provide users with a unified view of the data [15] whereas in data fusion, duplicates are merged and conflicting attributes values are identified and possibly repaired in order to provide a single consistent value for each data attribute [3]. Data fusion therefore involves duplicate detection, also known as *Entity Resolution* or record linkage, where the goal is to identify data records that refer to the same entity. The first step in a data integration or fusion system is identification of *linkage points* between the data sources, i.e., finding correspondences between attributes in the data sources. Traditionally, this is performed by *schema matching*, where the goal is to identify the schema elements of the various data sources that are semantically related. However, the massive growth in the amount of unstructured and semi-structured data in data warehouses and on the Web has created new challenges for this task. In what follows, we first describe an example real-world integration scenario and then describe the unique challenges not addressed in previous work.

Table 1: Data set statistics for the example scenario

Data Source	Records#	Fact#	Distinct Paths#
DBpedia	24,367	1.9 Million	1,738
Freebase	74,971	1.9 Million	167
SEC	1,981	4.5 Million	72

Consider a scenario where data about public companies is gathered from different sources on the Web. We have collected three data sets from online sources: Freebase [28], DBpedia [23], and the U.S. Securities and Exchange Commission (SEC) [29]. The data sets are respectively extracted using Freebase's Metaweb Query Language, DBpedia's SPARQL endpoint, and IBM SystemT [4] applied on the online SEC forms. Each of the three data sets is converted into a collection of JSON encoded records. Each JSON record is a tree representing various facts about a public company (an *entity*). Table 1 shows some statistics over these data sets. One can see that the three data sets are very different in structure. JSON trees in DBpedia have over 1700 different paths (or *attributes*), and describe 1.9 million *fact*s (or entity-attribute-value triples), while JSON trees in SEC have only 72 distinct paths, but describe 4.5 million facts. This shows that DBpedia contains very heterogeneous records, while records in the SEC data set have a more consistent structure. Another observation is that Freebase has over 74,000 JSON records describing 1.9 million facts, while SEC has only approximately 2,000 records, but describes 4.5 million facts. The SEC data contains much more elaborate records than Freebase.

The three data sets should have significant overlap due to the common topic (public companies). Despite the fact that the SEC data has the greatest structural regularity, this data does not subsume the other data sets, so alignment and use of these data sets together can provide more information than any single source. Toward data integration, we first need to identify *linkage points*, i.e., paths (attributes) in the JSON trees that are shared or related among the data sets and that are useful in identifying entities that can be linked (a formal definition is given in the next section). One possible approach is to apply schema matching algorithms [18] based on the schema information of the data sets. A purely schema-based matching algorithm would fail in many cases. For instance, DBpedia contains the labels `dbpedia:stockSymbol`, `dbpedia:stockTicker` and `dbpedia:tickerSymbol` for stock symbols.[1] Further investigation of the instances reveals that each of these three attributes in DBpedia actually contain only a single value, perhaps because the DBpedia extraction algorithm has been unable to extract the stock symbols from Wikipedia. So, the label name in this case does not reflect the data, and is not useful for matching. Moreover, matching Freebase and SEC based on the `ticker_symbol` and `stockSymbol` attributes results in ambiguous links (one company matched with more than one company on the other side). This happens because some (subsidiary) companies in Freebase share stock symbols with their parent companies. This shows that these stock symbol attributes are not as *strong* linkage points as one would expect.

Even if the schema labels are meant to be representative and can be used for matching schema elements, there could be differences in data representation and style that make matching the records difficult. In fact, our experience in this and other similar scenarios (as described in Section 5) show that for the most interesting and useful linkage points, schema labels and values do not match using simple string comparison. For example, there are different attribute labels used for URLs (e.g., `url` in Freebase, `foaf:page` and `foaf:homepage` in DBpedia) and there are different ways of writing URLs (e.g., `http://ibm.com` vs. `http://www.ibm.com/`). Another example is different representations of identifiers, e.g., the unique identifiers in SEC, called CIKs, are fixed-length numbers such as `#0000012345` stored in an attribute labeled `cik` but Freebase represents them as `/business/cik/12345` stored in the identifier attribute `id`. There are also cases where only a part of the values can be used to link the records. For example, URI `http://dbpedia.org/resource/Citigroup` in DBpedia matches with ID `/en/topic/citigroup` in Freebase, and URL `http://www.audi.com/` matches with name `Audi`. Since `Citigroup` and `Audi` are relatively rare names, (URI,ID) and (URL,name) attribute pairs can effectively be used to link these records. However, such linkage points can easily be missed unless the user has a thorough knowledge of both data sets. Gaining such knowledge could be challenging as it may require examining a large and representative portion of the data to understand when an attribute could be useful in linking a portion of the data.

In traditional data integration systems, identification of linkage points is performed either manually (possibly using a user-interface designed for matching schema elements) or by an automatic or semi-automatic *schema matching* algorithm. However, the size and heterogeneity of the schema, along with schema errors present in many sources that use automated information extraction, make many existing schema-based approaches inaccurate in aligning may data sets. In addition, the size and heterogeneity of Web data makes existing instance-based approaches [14, 22] ineffective and inefficient in aligning data.

[1]The data we have used was retrieved in January 2010. DBpedia data and schema have considerably changed since then, so the statistics and examples here may no longer be current, but can be verified using the data dumps available on our project page [30].

In this paper, we present a framework for identification of linkage points for multi-source Web data integration. Our framework includes a novel class of search algorithms to identify strong linkage points (that is, attributes that can be used to link entities across data sets) even when such attributes are weak schema matches. Importantly, we are specifically looking for attributes that help in identification of entities that can be linked. So unlike in schema matching, we are not interested in finding all corresponding attributes (for example, matching `color` and `colour`). As a result, our search can be much more focused. Our algorithms take advantage of 1) a library of lexical analyzers, 2) fast record-level and token-level inverted indices, 3) a library of similarity functions, and 4) a set of filtering strategies to filter false-positive results of the search. We have implemented and experimentally evaluated the framework in several real world Web data integration scenarios such as the one described above. We show the effectiveness of different components of the framework in discovering linkage points in these scenarios, and how the discovered linkage points can enhance the record linkage process.

Next, we present our problem definition. Section 3 presents our proposed framework, and Section 4 presents the details of the search algorithms for attribute selection and identification of linkage points. We present a thorough experimental evaluation of the search algorithms in Section 5. Sections 6, 7, and 8 conclude the paper with a summary of the results, brief overview of the related work, and a few interesting directions for future work.

2. PRELIMINARIES

Our framework is designed for use on any semistructured data format. We present a simple data model for representing tree-structured documents. We make use of a set LABELS of constants for representing attribute labels (in a tree, an attribute is named by one of these labels or a path of these labels), and a set VALUES of constants representing (atomic) attribute values. The data model DM and sets DICTIONARY and LIST are defined recursively.

- If $x \in$ VALUES, then $x \in$ DM
- If $l_1, l_2, \ldots, l_n \in$ LABELS are distinct labels, and $v_1, v_2, \ldots, v_n \in$ DM, then $\{l_1 : v_1, l_2 : v_2, \ldots, l_n : v_n\} \in$ DICTIONARY. All dictionaries are in DM (DICTIONARY $\subseteq$ DM).
- If $v_1, v_2, \ldots, v_n \in$ DM, then $[v_1, v_2, \ldots, v_n] \in$ LIST. All lists are in DM (LIST $\subseteq$ DM).
- Nothing else is in DM.

We refer to an element $x \in$ DM as a *document*.

We define FLAT to be an operator that maps nested lists to flat sets defined as:

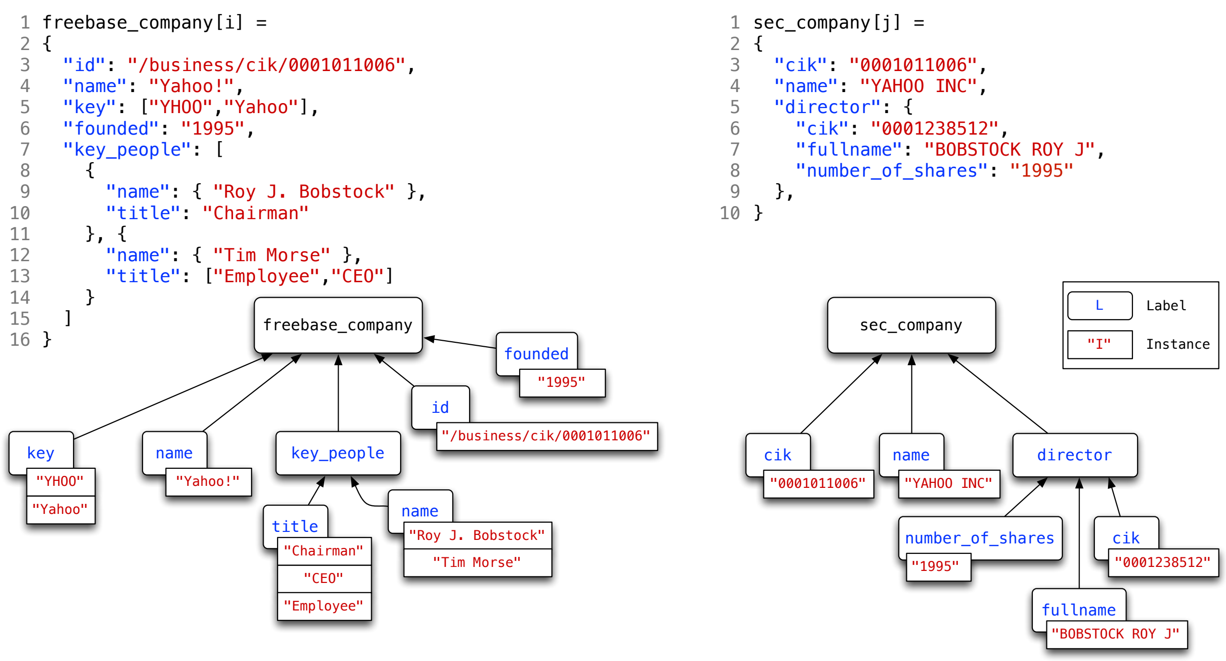
$$\textsc{Flat}(x) = \begin{cases} \{x\} & \text{if } x \notin \textsc{List} \\ \bigcup_i \textsc{Flat}(v_i) & \text{if } x = [v_1, v_2, \ldots, v_n] \end{cases}$$

If $x = \{l_1 : v_1, l_2 : v2, \ldots, l_n : v_n\}$, then we define $x[l_i] = v_i$, and $\textsc{Keys}(x) = \{l_1, l_2, \ldots, l_n\}$.

DEFINITION 1 (PATH AND EVALUATION). *A path p is a sequence of labels, written $p = \langle \texttt{root}, l_1, l_2, \ldots, l_n \rangle$ where $l_i \in$ LABELS. We write $p \cdot l_{n+1}$ to mean $\langle \texttt{root}, l_1, l_2, \ldots, l_n, l_{n+1} \rangle$. Given a document x, we define the evaluation of a path p, denoted by $[\![p|x]\!]$ as:*

$$\begin{aligned} [\![\texttt{root}|x]\!] &= \textsc{Flat}(x) \\ [\![p \cdot l|x]\!] &= \bigcup \{\textsc{Flat}(y[l]) : y \in [\![p|x]\!] \wedge l \in \textsc{Keys}(y)\} \end{aligned}$$

DEFINITION 2 (DATA SETS, ATTRIBUTES AND INSTANCES). *A data set, D, is defined as a collection of documents:*



```
freebase_company[i] =
{
  "id": "/business/cik/0001011006",
  "name": "Yahoo!",
  "key": ["YHOO","Yahoo"],
  "founded": "1995",
  "key_people": [
    {
      "name": { "Roy J. Bobstock" },
      "title": "Chairman"
    }, {
      "name": { "Tim Morse" },
      "title": ["Employee","CEO"]
    }
  ]
}
```

```
sec_company[j] =
{
  "cik": "0001011006",
  "name": "YAHOO INC",
  "director": {
    "cik": "0001238512",
    "fullname": "BOBSTOCK ROY J",
    "number_of_shares": "1995"
  },
}
```

Figure 1: Example records, paths, and their evaluations

$D = \{r_1, r_2, \ldots, r_n\}$ *where* $r_i \in$ DM. *We refer to* $r \in D$ *as the* records *of* D. *(These records are what we have informally called entities in the introduction.)*

An attribute *of* $r \in D$ *is a path,* p*, such that:*

$$[\![p|r]\!] \cap \textsc{Values} \neq \emptyset$$

We sometimes refer to an attribute $\langle \texttt{root}, l_1, \ldots, l_k \rangle$ *in data set* D *as* D `->` l_1 `->` $\ldots$ `->` l_k. *The set of all attributes of* r *in the data set is denoted as* $\text{Attr}_r(D)$. *Similarly,* $\text{Attr}(D) = \bigcup_{r \in D} \text{Attr}_r(D)$.

The instances *of attribute* p *in a record* r *are defined as* $\text{Instances}_r(p) = [\![p|r]\!] \cap \textsc{Values}$. *The instances of a data set* D *are* $\text{Instances}_D(p) = \bigcup_{r \in D} \text{Instances}_r(p)$.

We sometimes refer to the set of instances of a data set (or more formally record-attribute-instance triples) as the set of *facts*.

EXAMPLE 1. *Figure 1 shows two example records from two data sets along with their JSON representation. One record describes the company* `Yahoo!` *in the Freebase data, and the other describes the same company in the SEC data set. The* LABELS *set includes* {`"id"`, `"name"`, `"key"`, `"founded"`, `"key_people"`, `"title"`, `"cik"`, `"director"`, `"fullname"`, `"number_of_shares"`} *while the* VALUES *set includes* {`"/business/cik/0001011006"`, `"Yahoo!"`, `"YHOO"`, `"Yahoo"`, `"1995"`, `"Roy J. Bobstock"`, `"Chairman"`, `"Tim Morse"`, `"Employee"`, `"CEO"`, `"0001011006"`, `"YAHOO INC"`, `"0001238512"`, `"BOBSTOCK ROY J"`}. *In the first record,* `"key"`*:* `["YHOO", "Yahoo"]` *is a* LIST *while* {`"name"`*:* {`"Roy J. Bobstock"`}, `"title"`*:* `"Chairman"`} *is a* DICTIONARY. *An example path in the first record is* $p = \langle \texttt{root}, \texttt{key\_people}, \texttt{name} \rangle$. *The evaluation of this path given the first record is* {`"Roy J. Bobstock"`, `"Tim Morse"`}. *Therefore this path is also an attribute, which we also refer to as* `freebase_company -> key_people -> name`*, and its set of instances is* {`"Roy J. Bobstock"`, `"Tim Morse"`}. *The first record contains 10 instances (or facts) while the second record contains 5 instances.*

A *lexical analyzer* (or tokenizer) l is defined as a function that takes an atomic value v and converts it into a set of *tokens* $\mathbf{v}$. Some analyzers only split the string value into a set of tokens such as word tokens or q-grams (substrings of length q of the string). Other analyzers perform string transformations (or *normalization*) by for example removing specific characters or changing letter case in addition to tokenization (or without any tokenization). We refer to the set of tokens of all the instance values of attribute p in record r as $Instances_r^l(p)$. Similarly, $Instances_D^l(p)$ represents the set of all the tokens of all the instance values of attribute p in data set D.

A *record matching* function is defined as a Boolean function $f(r_s, r_t)$ that returns true if the two records r_s and r_t match, according to a matching criteria. The matching criteria can be defined using an *attribute matching* function $f_{(p_s,p_t)}(r_s, r_t)$, that returns true if the instance values of the two attributes $Instances_{r_s}(p_s)$ and $Instances_{r_t}(p_t)$ are *relevant*. Using a Boolean *relevance function* $r(V_s, V_t)$, we say that two sets of values V_s and V_t are relevant if $r(V_s, V_t)$ is true. There are several ways to define the notion of relevance. For example, two sets V_s and V_t can be considered relevant if there exists $v_s \in V_s$ and $v_t \in V_t$ such that v_s and v_t are *similar*. Two atomic values v_s and v_t are considered similar if their similarity score according to a value similarity function $sim()$ is above a threshold θ, i.e., $sim(v_s, v_t) \geq \theta$.

DEFINITION 3 (LINKAGE POINT). *We define a linkage point between two data sets* D_s *and* D_t *as a pair of attributes* (p_s, p_t) *such that for some attribute matching function* f*, the following set is non-empty:*

$$M_{(p_s,p_t)} = \{(r_s, r_t) | r_s \in D_s \wedge r_t \in D_t \wedge f_{(p_s,p_t)}(r_s, r_t)\} \quad (1)$$

M *is referred to as the* ***linkage set****.*

EXAMPLE 2. *In Figure 1, the attribute* $p_s = \langle \texttt{root}, \texttt{key\_people}, \texttt{name} \rangle$ *in the first record and attribute* $p_t = \langle \texttt{root}, \texttt{director}, \texttt{full\_name} \rangle$ *in the second record form a linkage point* (p_s, p_t) *using the attribute matching and relevance functions defined above, Jaccard similarity between lowercase word tokens as* $sim()$*, and threshold* $\theta = 0.4$ *since* $sim(\text{"Roy J. Bobstock"}, \text{"BOBSTOCK ROY J"}) = 2/4 = 0.5$.

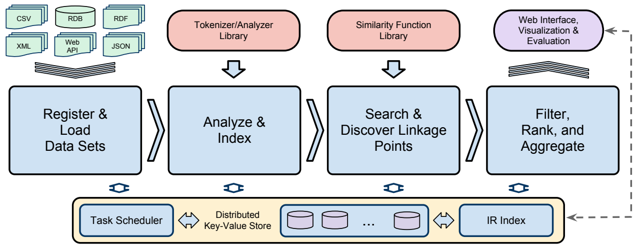

CSV
RDB
RDF
XML
Web API
JSON
Tokenizer/Analyzer Library
Similarity Function Library
Web Interface, Visualization & Evaluation
Register & Load Data Sets
Analyze & Index
Search & Discover Linkage Points
Filter, Rank, and Aggregate
Task Scheduler
Distributed Key-Value Store
IR Index

Figure 2: SMaSh framework

To align data, we need to find attributes containing values that when linked can help to identify related records (entities). To do this, we use two measures, *strength* and *coverage*. Given a potential linkage point (p_s, p_t), the *strength* measures how identifying the links between values of these attributes are. Strength is defined as the percentage of distinct records that can be linked by a pair of attributes. For example, in Figure 1, attribute `freebase_company -> founded` and `sec_company -> director -> number_of_shares` can be used to link the two records. However, if many other companies are founded in the same year or have the same number of shares, this linkage set will also link these records with several other records and therefore form a linkage point with low strength. The *coverage* measures how many records are linkable. While high coverage is better, strong linkage points that have good, but not full, coverage are still useful to link subsets of records.

DEFINITION 4 (STRENGTH & COVERAGE). *We define the strength of a linkage set as the percentage of distinct records that appear in the set, i.e.:*

$$strength(M_{(p_s,p_t)}) = \frac{|S_{(p_s,p_t)}| + |T_{(p_s,p_t)}|}{|M_{(p_s,p_t)}| \times 2} \quad (2)$$

where $S_{(p_s,p_t)} = \{s|\exists t : (s,t) \in M_{(p_s,p_t)}\}$ *and* $T_{(p_s,p_t)} = \{t|\exists s : (s,t) \in M_{(p_s,p_t)}\}$.

We define the coverage of a linkage set as the percentage of source and target records that appear in the set:

$$coverage(M_{(p_s,p_t)}) = \frac{|S_{(p_s,p_t)}| + |T_{(p_s,p_t)}|}{|D_s| + |D_t|} \quad (3)$$

3. FRAMEWORK

Our framework for linkage point discovery takes input data sets and returns a (ranked) set of linkage points. The discovery is performed in a scalable, online fashion, suitable for large Web data sets, and is illustrated in Figure 2.

Task Scheduler, Storage and Indexing Backend. The backend includes a task scheduler that manages (Web) data retrieval, indexing, and discovery tasks, in addition to fast memory and disk-based key-value store, and indexing engines. All the tasks are performed in a way that at any point, partial results can be made available to users of the system. The task scheduler prioritizes tasks and keeps track of their status. Depending on the size of the data, index, and the type of the discovery process, the memory-based key-value store can be used or can be replaced by a disk-based index.

Register and Load Data Sets. This component allows users to *register* a wide variety of data sets. The input could be in XML, RDF (XML, N3 or NTriples), JSON, relational, or CSV format. The data can also come directly from a set of popular Web APIs including those that support SPARQL or the Freebase API. Users can then *load* data, which will transform their data into our custom JSON format for representing data sets, store it locally, and create basic statistics to help optimize linkage point discovery. Note that our techniques will apply to Web sources that publish data sets of records representing entities of a single type (that is, one data set could be a set of companies, another could be a set of clinical trials, etc.) Furthermore, we make the assumption that each data set represents a set of distinct entities (with few duplicates). This assumption is commonly met by most Web sources which are generally curated. Of course if it is not, we could apply a pre-processing deduplication on each source.

Analyze and Index. This component constructs the set of attributes (as defined in Section 2) and indexes the attribute values using one of the following analyzers available in the *lexical analyzer* library of the system.

- *Exact analyzer* does not make any change to the string.
- *Lower analyzer* turns the string value into lowercase.
- *Split analyzer* breaks the string into a set of tokens by splitting them by whitespace after using the lower analyzer. E.g., "IBM Corp." turns into two terms "ibm" and "corp.".
- *Word token analyzer* first replaces all the non-alphanumeric characters with whitespace and then uses the split tokenizer to tokenize the string. E.g., "http://ibm.com" is tokenized into terms "http", "ibm" and "com".
- The *q-gram analyzer* tokenizes the string into the set of all lowercase substrings of length q. It also replaces all whitespaces with $q-1$ occurences of a special character such as \$. E.g., a 3-gram tokenizer will tokenize "IBM Corp." into "\$\$i", "\$ib","ibm","bm\$","m\$\$","\$\$c","\$co", "cor","orp","rp.","p.\$" and ".\$\$".

Note that the analyzer library can be extended with other domain-specific analyzers based on the application.

Search and Discover Linkage Points. This component searches for linkage points. The search uses a similarity function over the values indexed in the previous component using one of the available analyzers. We use a library of common similarity functions that includes the following.

- *Intersection* similarity function returns the intersection size of the two value sets. Given two value sets V_1 and V_2:

$$intersect(V_1, V_2) = |V_1 \cap V_2| \quad (4)$$

- *Jaccard* returns the Jaccard coefficient of the two value sets:

$$jaccard(V_1, V_2) = \frac{|V_1 \cap V_2|}{|V_1 \cup V_2|} \quad (5)$$

- *Dice* returns the Dice coefficient of the two value sets:

$$dice(V_1, V_2) = \frac{2|V_1 \cap V_2|}{|V_1| + |V_2|} \quad (6)$$

- *MaxInc* returns the maximum inclusion degree of one value set in another:

$$maxinc(V_1, V_2) = \frac{|V_1 \cap V_2|}{min(|V_1|, |V_2|)} \quad (7)$$

The similarity library also includes functions that quantify the similarity of a (single) value string to the values in a set of instance values. These functions are based on well-established relevance functions (with highly efficient implementations) in information retrieval, such as Cosine similarity with tf-idf [20] and the Okapi BM25 [19]. These relevance functions are used to quantify the relevance of a query to a document. We use the same approach to determine the closeness of a value string $v_1 \in V_1$ to a value string $v_2 \in V_2$. For brevity, we describe only one such relevance function, the BM25 measure which is considered the state-of-the-art in document retrieval systems [19], though our system includes of a library of these functions.

- *BM25* similarity score is defined as follows:

$$bm25(v_1, v_2) = \sum_{t \in \mathbf{v}_1 \cap \mathbf{v}_2} \hat{w}_{v_1}(t) \cdot w_{v_2}(t) \quad (8)$$

where $\mathbf{v}_1$ and $\mathbf{v}_2$ are the set of tokens in v_1 and v_2 (tokenized using one of the above-mentioned analyzers), and:

$$\begin{aligned} \hat{w}_{v_1}(t) &= \frac{(k_3+1) \cdot tf_{v_1}(t)}{k_3 + tf_{v_1}(t)} \\ w_{v_2}(t) &= \log\left(\frac{N - n_t + 0.5}{n_t + 0.5}\right) \frac{(k_1+1) \cdot tf_{v_2}(t)}{K(v_2) + tf_{v_2}(t)} \\ K(v) &= k_1\left((1-b) + b\frac{|v|}{avg_{vl}}\right) \end{aligned}$$

and $tf_v(t)$ is the frequency of the token t in string value v, $|v|$ is the number of tokens in v, avg_{vl} is the average number of tokens per value, N is the total number of values in the set of values V_2, n_t is the number of records containing the token t and k_1, k_3 and b are a set of independent parameters [19].

Filter, Rank and Aggregate. Our search algorithm uses the above measures to create a set of candidate linkage points that is then filtered, ranked and aggregated. Filtering can be performed using several measures, for example:

- *Cardinality filter* removes a linkage point (p_s, p_t) if the cardinality of its corresponding linkage set $M_{(p_s,p_t)}$ is small, i.e., $|M_{(p_s,p_t)}| < \theta_l$ where θ_l is a user-defined cardinality threshold. A low cardinality of the linkage set may indicate an accidental match, or a linkage point that results in a very small number of links and therefore may not be desired depending on the application.
- *Coverage filter* uses the coverage measure from Definition 4 to remove a linkage point (p_s, p_t) if the coverage of its linkage set is less than a user-defined threshold θ_c, i.e., $coverage(M_{(p_s,p_t)}) < \theta_c$. The user selects a higher coverage threshold if it is known in advance that the set of entities the two data sets describe have a high overlap, and so a high coverage can indicate a high recall in the final linkage results.
- *Strength filter* uses the strength measure from Definition 4 to remove a linkage point (p_s, p_t) if $strength(M_{(p_s,p_t)}) < \theta_s$ where θ_s is a user-defined strength threshold. A high strength threshold can guarantee a higher precision in the final linkage results, although it may lower the recall depending on how clean each source is, and the data characteristics that may result in accidental matches.

The framework also includes a Web interface for task management and monitoring, visualization of the results and evaluation based on user feedback. Such results can be used, for example, to define entity resolution rules in the user's preferred language such as LinQL [10], Dedupalog [1], or HIL [11].

4. SEARCH ALGORITHMS

In this section, we present the details of our proposed search algorithms for linkage point discovery. The search process starts from a source data set, and goes through all its attributes. For each attribute, the set of instances (or a sample of this set) is retrieved from the index. Then the target data set is searched for similar values using a similarity function.

Algorithm 1: SMaSh-S

```
Input  : Data sets $D_s$ and $D_t$,
         A set similarity function $f$,
         A lexical analyzer $l$,
         A matching threshold $\theta$
Output: (Ranked) list of pairs of paths $(p_s, p_t)$
1 foreach attribute $p_s$ in $D_s$ do
2     foreach attribute $p_t$ in $D_t$ do
3         $score(p_s, p_t) \leftarrow f(Instances^l_{D_s}(p_s), Instances^l_{D_t}(p_t))$
4     end
5 end
6 return pairs of $(p_s, p_t)$ with $score(p_s, p_t) \geq \theta$ (in descending order of
  $score(p_s, p_t)$)
```

4.1 SMaSh-S: Search by Set Similarity

Algorithm 1 shows our simplest realization of the search strategy, which we refer to as the SMaSh-S algorithm. This algorithm goes through the set of attributes in the source data, and for each attribute p_s in the source data set D_s retrieves the set of instance values $Instances_{D_s}(p_s)$ and calculates the similarity of this set with all the instance value sets in D_t using the given set similarity function. The result is all pairs of attributes (p_s, p_t) that have similarity score above the given threshold θ, along with the value of the similarity score. A higher similarity threshold θ results in higher precision but lower recall.

We can create a linkage set $M_{(p_s,p_t)}$ for all the returned pairs by Algorithm 1 using an attribute matching function that returns true if the intersection of $Instances^l_{r_s}(p_s)$ and $Instances^l_{r_t}(p_t)$ is non-empty. Therefore, all the pairs returned by Algorithm 1 are linkage points (as defined in Definition 3).

Assuming that $f()$ is one of the set similarity functions discussed in Section 3, the average complexity of this algorithm is $|\text{Attr}(D_s)| \times |\text{Attr}(D_t)| \times \text{Average}_{p \in D}(|Instances^l_D(p)|)$ where $D = D_s \cup D_t$ for similarity computation, plus $|\text{Attr}(D_s)| \times |\text{Attr}(D_t)| \times log(|\text{Attr}(D_s)| \times |\text{Attr}(D_t)|)$ for sorting the results.

4.2 SMaSh-R: Record-based Search

The SMaSh-S algorithm can effectively find linkage points for data sets whose (tokenized) instance values have a sufficient overlap, and very few of the overlapping sets of values are accidental, i.e., only a few instance values are shared between unrelated attributes. Therefore, the performance of the algorithm is highly affected by the choice of lexical analyzer. Using simpler analyzers (such as *exact* or *lower*)) can result in high-quality linkage points only if the data sets follow the same style and formats (something that is not very common on the Web), and more complex analyzers (such as q-gram based tokenizers) can result in a large number of accidental matches.

Algorithm 2 shows an alternative search strategy that we refer to as the SMaSh-R algorithm. This algorithm first picks a sample of values (analyzed using analyzer l) for each source attribute. For each value in the sample set, it then searches the target data set for the top k similar values (i.e., values with value similarity score above a user-defined threshold τ using a string similarity function $sim()$). This search (Lines 4-6 in Algorithm 2) can be performed very efficiently using proper indices and state-of-the-art keyword search algorithms as described in Section 3. The algorithm then takes the average of the similarity score for each attribute in the result among all the record-attribute-value triples returned. If this score is above a user-defined threshold θ, the attribute pair will be returned as a linkage point.

For each attribute pair in the output of Algorithm 2, one can derive a non-empty linkage set using an attribute matching func-

Algorithm 2: SMaSh-R

```
Input  : Data sets $D_s$ and $D_t$,
         A lexical analyzer $l$,
         A value similarity function $sim()$,
         Value similarity threshold $\tau$,
         Value of $k$ for top-$k$ search,
         Sample value set size $\sigma_v$,
         A matching threshold $\theta$
Output: (Ranked) list of pairs of attributes $(p_s, p_t)$
1  foreach attribute $p_s$ in $D_s$ do
2      $Query\_Set \leftarrow$ { Up to $\sigma_v$ random values in $Instances^l_{D_s}(p_s)$}
3      foreach value $q$ in $Query\_Set$ do
4          $M \leftarrow \{(r_t, p_t, v) | p_t \in Attr_{r_t}(D_t) \wedge$
               $v \in Instances^l_{r_t}(p_t) \wedge sim(v, q) \geq \tau\}$
5          $M_{topk} \leftarrow \{(r_t, p_t, v) \in M$ with top $k$ highest $sim(v, q)\}$
6          $MS \leftarrow$ Multiset $\{(p_t, sim(v, q)) | \exists r_t : (r_t, p_t, v) \in M_{topk}\}$
7      end
8      $score(p_s, p_t) \leftarrow$ Average $sim$ value for all $(p_t, sim) \in MS$
9  end
10 return pairs of $(p_s, p_t)$ with $score(p_s, p_t) \geq \theta$ (in descending order of
   $score(p_s, p_t)$)
```

tion that returns true if at least one value v_s in $Instances_{r_s}(p_s)$ and one value v_t in $Instances_{r_t}(p_t)$ have similarity score $sim(v_s, v_t) \geq \tau$. Therefore, all the pairs returned by Algorithm 2 are linkage points (as defined in Definition 3).

The average complexity of this algorithm is $|\text{Attr}(D_s)| \times (\sigma_v \times (O(lookup) + k) + log(|\text{Attr}(D_t)|)$ where $O(lookup)$ is the average complexity of a probe in the search index. This algorithm is clearly more efficient than SMaSh-S assuming that the search index is built in a preprocessing phase. In Section 5, we show that small samples (that is, small values of $\sigma_v \ll |\text{Attr}(D_t)|$) can work very well in practice.

4.3 SMaSh-X: Improving Search by Filtering

The strategy used in the SMaSh-R algorithm can improve the search performance especially for more complex analyzers that are an important part of an effective search strategy. However, low-quality results (irrelevant pairs returned as linkage points) are unavoidable when dealing with highly heterogeneous data. Therefore, we propose an effective filtering strategy to improve the accuracy of the linkage points returned by the SMaSh-S and SMaSh-R algorithms. We refer to the resulting algorithm as the SMaSh-X algorithm. The filtering portion of the algorithm is shown in Algorithm 3. This part of the algorithm takes as input a set of linkage points along with their scores from the previous algorithms, creates a carefully selected subset of linkage sets for each linkage point, and filters out the linkage points whose linkage (sub-)sets do not have a user-defined *strength*, *coverage*, or *cardinality*.

A key issue in Algorithm 3 is effective and efficient creation of a subset of the linkage set that can effectively be used to prune irrelevant pairs from the results. For each pair (p_s, p_t), the algorithm first picks a sample set of size σ_s out of all the instances $Instances^l_{D_s}(p_s)$. Then, for each value in the sample set, the set of records containing the value (in attribute p_s) are looked up. This lookup can efficiently be performed using a reverse index built during the indexing phase. Then the set of matching records in the target data set are retrieved using the attribute matching function. For attribute pairs found using the SMaSh-S algorithm, the attribute matching function is a Boolean function that returns true if the intersection of $Instances^l_{r_s}(p_s)$ and $Instances^l_{r_t}(p_t)$ is non-empty. For attribute pairs found using the SMaSh-R algorithm, the matching function is a Boolean function that returns

Algorithm 3: SMaSh-X filtering

```
Input  : Data sets $D_s$ and $D_t$,
         List $L$ of linkage points with their score $(p_s, p_t)$
         with corresponding attribute matching function
         $f()$ and lexical analyzer $l$,
         Sample linkage set size $\sigma_s$,
         Cardinality threshold $\kappa$,
         Smoothing cutoff limit $\lambda$,
         Coverage threshold $\chi$,
         Strength threshold $\tau$,
         Matching threshold $\theta$
Output: (Ranked) list of pairs of paths $(p_s, p_t)$
1  foreach pair $(p_s, p_t) \in L$ do
2      $M_{(p_s,p_t)} \leftarrow \emptyset$
3      $V \leftarrow$ Sample of size $\sigma_s$ of $Instances^l_{D_s}(p_s)$
4      foreach $v \in V$ do
5          $R_s \leftarrow \{r_s \in D_s | v \in Instances^l_{r_s}(p_s)\}$
6          Construct $r_s$ with path $p_s$ and value $v$ such that
           $Instances^l_{r_s}(p_s) = \{v\}$
7          $R_t \leftarrow \{r_t \in D_t | f_{(p_s,p_t)}(r_s, r_t)\}$
8          if $|R_s| > \lambda$ then
9              $R_s \leftarrow$ subset of size $\lambda$ of $R_s$
10         end
11         if $|R_t| > \lambda$ then
12             $R_t \leftarrow$ subset of size $\lambda$ of $R_t$
13         end
14         $M_{(p_s,p_t)} \leftarrow M_{(p_s,p_t)} \cup \{(r_s, r_t) | r_s \in R_s \wedge r_t \in R_t\}$
15     end
16     if $strength(M_{(p_s,p_t)}) > \tau$ and $coverage(M_{(p_s,p_t)}) > \chi$
17         and Cardinality of both $p_s$ and $p_t$ are above $\kappa$
18     then
19         $score'(p_s, p_t) \leftarrow score(p_s, p_t) \times strength(M_{(p_s,p_t)})$
20     end
21 end
22 return pairs of $(p_s, p_t)$ with $score'(p_s, p_t) \geq \theta$ (in descending order of
   $score'(p_s, p_t)$)
```

true if at least one value v_s in $Instances_{r_s}(p_s)$ and one value v_t in $Instances_{r_t}(p_t)$ have similarity score $sim(v_s, v_t) \geq \tau$. The matching records can be retrieved and saved (cached) while performing the SMaSh-S or SMaSh-R algorithms to speed up the filtering operation.

When the matching source/target record pairs are found, they are added to the linkage set $M_{(p_s,p_t)}$, but we limit the size of each set of matching records to a user-defined limit λ. This can be seen as a way to *smooth* the strength measure, in order to decrease the effect of those infrequent values that match with a very large number of records, such as a non-standard null value representation (for example, `None` or `N/A`) that are very common in Web data.

When the linkage set sample is created, its strength, coverage and cardinality are checked and the pair is omitted from the results if any of the values are below the thresholds. Finally, the score is adjusted by the strength measure to lower the rank of those linkage points that have lower strength, and potentially remove them if the adjusted score drops below the matching threshold θ.

The worst case complexity of the algorithm is $|L| \cdot \log |L| \times \sigma_s \times \lambda^2$ assuming that the lookup in Lines 5 and 7 can be performed in $O(1)$ by indexing (caching) the similarity results while generating L. We show in Section 5 that a small sample size σ_s is sufficient in practice. A small constant λ value results in both better performance and accuracy. The user needs to pick the right value of input parameters cardinality threshold κ, coverage threshold χ, and strength threshold τ depending on data characteristics such as the amount of noise in the input data. For example, the strength threshold can be lowered to accommodate for data sets that are not very

Table 2: Data set statistics

Entity	Source	Data Set	Rec#	Fact#	Attr#
Company	Freebase	`fbComp`	74,971	1.92M	167
	SEC	`secComp`	1,981	4.54M	72
	DBpedia	`dbpComp`	24,367	1.91M	1,738
Drug	Freebase	`fbDrug`	3,882	92K	56
	DrugBank	`dbankDrug`	4,774	1.52M	145
	DBpedia	`dbpDrug`	3,662	216K	337
Movie	Freebase	`fbMovie`	42,265	899K	57
	IMDb	`imdbMovie`	14,405	483K	41
	DBpedia	`dbpMovie`	15,165	1.57M	1,021

clean and may contain duplicates. The values of χ and κ should be increased if the application requires a large coverage, and lowered if a high recall in the record linkage process is needed. We discuss automatic selection of parameters in Sections 5.5 and 6.

5. EXPERIMENTS

In this section, we present the results of evaluating the SMaSh algorithms in real-world integration scenarios.

5.1 Data Sets

Table 2 shows the statistics of the data sets for three real-world integration scenarios. All these scenarios are similar to our motivating example described in Section 1, where the goal is to discover linkage points that can be used to perform entity resolution. In other words, we are looking for linkage points whose corresponding linkage sets (or a significant part of the linkage sets) consist of record pairs that refer to the same real-world entity.

Each scenario involves resolution of a single entity type, and includes three data sources. Two data sources Freebase [28] and DBpedia [23] are shared in all the scenarios. As described earlier, the Freebase data is downloaded in JSON format using a query written in Metaweb Query Language (MQL) to fetch all the attributes related to each entity type. DBpedia data is fetched from DBpedia's SPARQL endpoint in RDF/N-Triples format and converted into JSON. Company scenario uses data extracted from the U.S. Securities and Exchange Commission (SEC) online form files using IBM's SystemT [4] with output in JSON format. The drug scenario uses information about drugs extracted from DrugBank [24], which is an online open repository of drug and drug target data. Movie scenario uses movie data from the popular online movie database, IMDb [25]. IMDb contains a huge number of movies, TV shows and episodes, and even video games, from all around the world. For our experiments in this section, we pick a sample of movies that are most likely to appear on DBpedia or Freebase, to make sure that the data sets overlap. Our system is capable of automatically picking a high-quality sample of a data source to guarantee overlap with other sources, the details of which are beyond the scope of this paper.

5.2 Settings & Implementation Details

The framework shown in Figure 2 is implemented using a number of state-of-the-art storage, indexing, and task management systems. As our storage engine, we use Redis [26], which is an open-source networked, in-memory key-value store with optional disk-based persistence. We use Redis as our storage backend and as the backend for indexing of all the value sets for efficient set similarity computation. We use separate Redis servers to store the original data sets, meta-data and task information, and reverse indices for each lexical analyzer. We experiment with seven different analyzers, namely *exact*, *lower*, *split*, *word token*, *2-gram*, *3-gram*, and *4-gram* analyzers. This results in nine Redis server instances (two for data and meta-data, seven for analyzers). For our run-time experiments, we ran the Redis instances for each scenario on one of three servers each with an Intel X3470 Xeon Processor, and 24GB of RAM. For a fair comparison, we have made sure that the Redis instances for each experiment fully fit in memory and no swapping occurs. For the SMaSh-R algorithm and implementation of our search index for the $BM25$ similarity measure, we use Xapian [27] search engine library that includes a built-in implementation of the $BM25$ function along with a highly efficient disk-based index.

Table 3: Summary of algorithm parameters

Algorithm	Par.	Description	Default
All	θ	Matching threshold	0.5
SMaSh-R	τ_v	Value similarity threshold	0.5
	k	Value of k for top-k search	200
	σ_v	Sample value set size	$20,000$
SMsSh-X	τ_s	Strength threshold	0.6
	κ	Cardinality threshold	30
	χ	Coverage threshold	0.001
	λ	Smoothing cutoff limit	50
	σ_s	Sample value set size to build linkage set	$1,000$

A summary of the input parameters of the system is shown in Table 3. Unless specifically noted, we use the default parameter values in Table 3. Note that these default values are ones that one would naturally choose as a reasonable value for an initial experimentation. We have not performed any learning or experimentation with a specific data set to derive these values. In the next section, we also report on the effect of changing these values for each of the algorithms and data sets.

5.3 Accuracy Results

For the scenarios in this paper, we consider a linkage point to be *relevant* if it consists of attributes that are semantically related *and* it can be used to perform entity resolution, i.e., (a significant portion of) its corresponding linkage set consists of record pairs that refer to the same real-world entity. To find the ground truth, we went through the daunting task of examining hundreds of linkage points for each scenario that were found using several different settings of each of the algorithms (overall 336 sets of results for each scenario). We constructed the linkage sets for each of the linkage points and went through a sample to verify that a large number of records in the linkage sets refer to the same real-world entity. We also manually inspected a sample of matching records (e.g., records that represent `IBM` in Freebase, DBpedia and SEC) to make sure there are no linkage points that none of our algorithms can identify. Table 9 shows some example linkage points in the company scenario that have been found. The full ground truth along with the data sets and the details of all the results obtained are available on our public project web page [30].

5.3.1 Measures

To evaluate the accuracy of the results, we use well-known measures from IR, namely precision, recall, F measure, reciprocal rank, and fallout. *Precision* is the percentage of the linkage points in the output that are relevant. *Recall* is the percentage of the relevant linkage points in the output of the algorithm. The F measure is the weighted harmonic mean of precision and recall, i.e.,

$$F_\beta = \frac{1 + \beta^2}{\frac{\beta^2}{Re} + \frac{1}{Pr}} \quad (9)$$

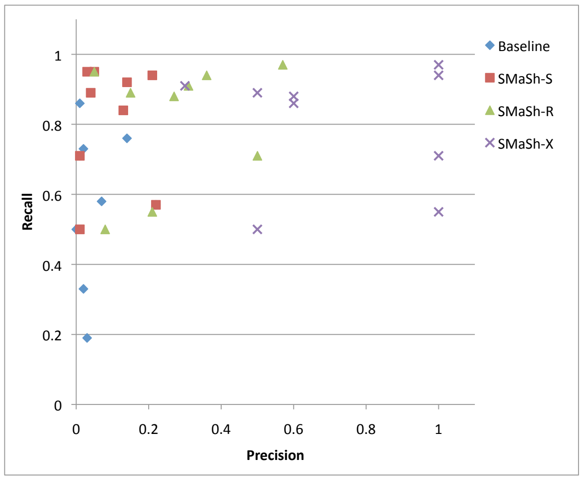


Figure 3: Best precision/recall values for all scenarios

We report the F_2 scores that weights recall twice as much as precision since in our scenarios, a low recall value negatively affects the ability to perform entity resolution whereas a low precision (and a high recall) only results in more linkage points to be examined for relevance. We also report *Reciprocal Rank (RR)* which is defined as the multiplicative inverse of the rank of the first correct answer. RR measures the ability of the algorithm to return a relevant result to the user high in the ranked results. *Fallout* is the percentage of all the possible non-relevant results that are returned in the output. A high fallout score indicates too many irrelevant results in the output and therefore poor performance of the algorithm.

5.3.2 *Evaluation Results*

We have thoroughly evaluated all the algorithms using all the above mentioned analyzers and similarity functions, and a wide range of parameters. Here, we only report our most interesting observations. First, we report the highest accuracy results achieved by each of the algorithms in each of the scenarios. The highest accuracy is the best result obtained for each algorithm over all possible choices of analyzers and similarity functions (we discuss how to choose the right settings in Section 5.5). As a baseline, we use the SMaSh-S algorithm with intersection as the similarity function along with exact or lower analyzers. This setting of the SMaSh-S algorithm resembles the case where a user tries to manually examine the intersection of all the values for all the attributes by issuing standard queries over Web APIs or an RDBMS.

Table 4 and Figure 3 show the best accuracy results achieved for each SMaSh algorithm in each of the scenarios using different analyzers and similarity functions. The second column in Table 4 shows the number of linkage points in the ground truth, which varies from 2 (in `fbMovie` vs. `imdbMovie` scenario where the sources are very different in structure and values) to 76 (in `fbComp` vs. `dbpComp` scenario where the two data sets share a large number of values that are extracted from Wikipedia). Note that in Table 4, the best precision (and recall) reported may be from a setting different than that of the best F_2. Also note that in Figure 3, each data point may be from a different setting that achieves the best precision or recall, and so is different from a traditional precision-recall curve. The results show that in the majority of the scenarios, SMaSh-X outperforms the other algorithms, whereas SMaSh-R outperforms SMaSh-S and all the algorithms significantly improve the baseline performance.

Table 5 shows the highest F_2 score achieved in each scenario along with the analyzer and similarity function that achieved this score. The SMaSh-X algorithm has performed best in all the scenarios, although different analyzers and similarity functions result in the highest F_2. This shows that as expected, there is no single analyzer/similarity function that can work well in all scenarios. However, intersection and BM25 similarity functions have performed best along with exact, lower and word token analyzers. In some scenarios, the highest accuracy result obtained is relatively low. For example, in the `fbMovie-dbpMovie` scenario, the best F_2 obtained is 0.56 and the maximum precision is only 0.30. The reason behind this low accuracy is again the inability of one single strategy to find all the linkage points even in a single scenario. In the `fbMovie-dbpMovie` data, there are attributes that contain URLs of movies on which a q-gram or word token analyzer is not suitable whereas there are attributes containing movie titles that require these analyzers.

Figure 4 shows the effect of sample size value σ_v on the SMaSh-X algorithm for the three company data scenarios, using word token analyzer and BM25 similarity function. The effect is highly similar for other analyzers and scenarios. As shown in the figure, very low sample size values below 250 will result in poor quality and in some cases reasonable results which only shows small sets of random values can sometimes be used to find high-quality linkage points. On the other hand there is no significant change after $\sigma_v =$ 1000 for most cases, which shows the effectiveness of using only a sample of values in the SMaSh-R (and SMaSh-X) algorithm.

Figure 5 shows the effect of each of the filtering methods of the SMaSh-X algorithm on the maximum F_2 score achieved using different settings (shown in the same order as Table 5). All the filters are effective in all the scenarios (except for one), but each filter be-

Table 5: Best accuracy results for each scenario

Scenario		Best F_2		
ds1	ds2	analyzer	sim	F_2
`fbComp`	`secComp`	wordsplit	BM25	0.83
`fbComp`	`dbpComp`	lower	intersect	0.55
`dbpComp`	`secComp`	wordsplit	BM25	0.61
`fbDrug`	`dbankDrug`	exact	maxinc	0.71
`fbDrug`	`dbpDrug`	wordsplit	BM25	0.65
`dbpDrug`	`dbankDrug`	wordsplit	BM25	0.62
`fbMovie`	`imdbMovie`	exact,lower	intersect	0.5
`fbMovie`	`dbpMovie`	exact	intersect	0.56
`dbpMovie`	`imdbMovie`	split	BM25	0.70

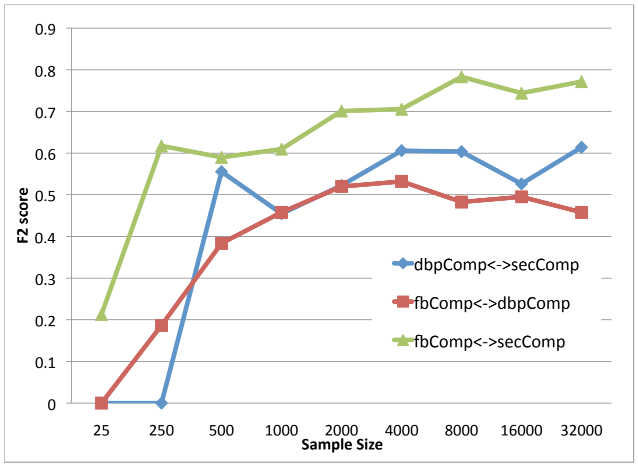


Figure 4: Effect of sample size σ_v on accuracy

Table 4: Best accuracy results of the SMaSh algorithms

Scenario			Best Precision				Best Recall				Best F_2				Best Fallout				Best Reciprocal Rank			
ds1	ds2	L.P. #	Baseline	SMaSh-S	SMaSh-R	SMaSh-X	Baseline	SMaSh-S	SMaSh-R	SMaSh-X	Baseline	SMaSh-S	SMaSh-R	SMaSh-X	Baseline	SMaSh-S	SMaSh-R	SMaSh-X	Baseline	SMaSh-S	SMaSh-R	SMaSh-X
`fbComp`	`secComp`	36	0.03	0.13	0.57	1.00	0.19	0.84	0.97	0.97	0.09	0.40	0.68	0.83	0.03	0.02	0.00	0.00	0.33	1.00	1.00	1.00
`fbComp`	`dbpComp`	76	0.22	0.22	0.21	1.00	0.57	0.57	0.55	0.55	0.43	0.43	0.42	0.55	0.00	0.00	0.00	0.00	1.00	1.00	1.00	1.00
`dbpComp`	`secComp`	13	0.02	0.04	0.15	0.50	0.33	0.89	0.89	0.89	0.06	0.17	0.43	0.61	0.00	0.00	0.00	0.00	0.01	1.00	0.25	1.00
`fbDrug`	`dbankDrug`	17	0.14	0.21	0.36	1.00	0.76	0.94	0.94	0.94	0.40	0.50	0.52	0.71	0.01	0.01	0.00	0.00	1.00	1.00	1.00	1.00
`fbDrug`	`dbpDrug`	26	0.07	0.14	0.27	0.60	0.58	0.92	0.88	0.88	0.25	0.36	0.48	0.65	0.01	0.01	0.00	0.00	0.33	1.00	1.00	1.00
`dbpDrug`	`dbankDrug`	24	0.01	0.05	0.31	0.60	0.86	0.95	0.91	0.86	0.07	0.20	0.47	0.62	0.03	0.01	0.00	0.00	1.00	1.00	0.50	1.00
`fbMovie`	`imdbMovie`	2	0.00	0.01	0.08	0.50	0.50	0.50	0.50	0.50	0.02	0.03	0.25	0.50	0.09	0.08	0.00	0.00	0.25	0.25	0.11	1.00
`fbMovie`	`dbpMovie`	22	0.02	0.03	0.05	0.30	0.73	0.95	0.95	0.91	0.09	0.14	0.19	0.56	0.01	0.01	0.00	0.00	1.00	1.00	0.14	1.00
`dbpMovie`	`imdbMovie`	8	0.01	0.01	0.50	1.00	0.71	0.71	0.71	0.71	0.04	0.07	0.41	0.69	0.01	0.01	0.00	0.00	0.20	0.50	1.00	1.00

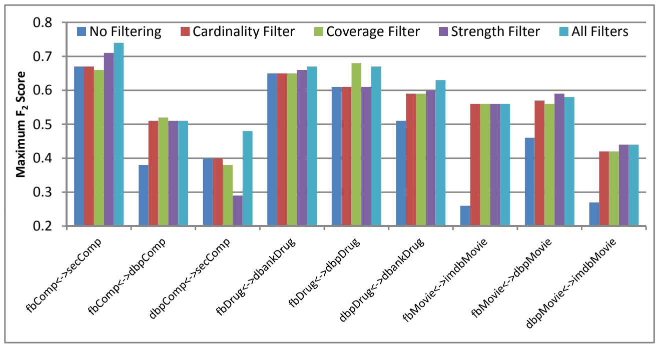


Figure 5: Effect of different filters on accuracy

haves differently depending on the scenario. On the other hand, the combination of all the filters performs consistently and relatively comparably to individual filters.

5.4 Running Times

Table 6 shows the load and indexing times of the nine scenarios. The first column shows the time to load the data from a file, flatten the structure and index the facts. This time is linear in the number of the facts in the data set (see Table 2). Note that our system is also able to fetch facts directly from Web APIs. The rest of the columns show the time it takes to index instance value sets both for SMaSh-S (set similarity calculation) and the IR index for BM25 similarity, using different analyzers. The time shown in the second column also includes the time it takes to build the reverse index for values required for building linkage sets for the SMaSh-X algorithm. This time is also linear in the size of the facts, although the complete task is clearly much more expensive than just indexing facts. The only exception is the `secComp` data set which has indexing time comparable to loading time. This is due to the fact that the data is automatically extracted from documents and contains a large number of instance values which are repeated in each record. Since we index value sets, the number of values indexed is much smaller (788K) than the number of facts (4.54M). Overall, these times show the scalability of the framework given that loading and indexing is a one-time process, and more importantly, can be done in an incremental fashion. As described in Section 3, our system supports running the indexing in the background and gradually updating the results.

The running time for the SMaSh-S algorithm across all the scenarios, analyzers, similarity functions, and parameters in our experiments was 22.93 seconds on average, with minimum 1.22 seconds, maximum 119.37, and 23.95 seconds standard deviation. For the SMaSh-R algorithm, the average was 338.08 seconds, with a minimum 14.58, maximum 3648.63, and 451.18 seconds standard deviation. The time for the filtering part of the SMaSh-X algorithm varied between 0.69 and 1062.59 seconds with average 138.00 and 246.54 standard deviation. Table 7 shows the running time of the best-performing algorithms in each scenario (maximum F_2 values shown in Table 5). Note that the SMaSh-R algorithm is considerably slower than the SMaSh-S algorithm in our implementation due to memory-based implementation of set similarity measures and disk-based IR index. As the size of the data and index increase beyond the memory size, we expect SMaSh-R to outperform the SMaSh-S algorithm. It is important to note that these are the times to complete running the algorithms, although our framework can return partial results while the data sets are being loaded, indexed, and matched. In our experience, our system can return a reasonable number of linkage points in all the scenarios and settings in just a few seconds.

Table 6: Load and indexing time (seconds)

data set	load	exact+	lower	split	word	3-gram
fbComp	117.33	1,277.52	514.99	725.13	894.96	4,109.21
dbpComp	113.31	1,081.80	904.93	1,775.87	3,109.40	19,420.09
secComp	289.61	453.49	287.79	326.99	423.54	2,050.25
fbDrug	5.13	30.58	27.29	29.97	41.55	216.81
dbpDrug	12.39	92.25	70.40	113.99	250.28	1,441.49
dbankDrug	108.36	640.72	391.13	853.64	975.80	8,669.88
fbMovie	62.92	347.91	253.97	266.80	388.03	2,123.57
dbpMovie	93.03	685.64	528.76	926.44	1,680.56	9,835.89
imdbMovie	32.01	407.62	255.51	409.96	453.19	2,513.92

Table 7: Running time of the best performing algorithm for each scenario

Scenario		time (seconds)		
ds1	ds2	analyzer	sim	time
`fbComp`	`secComp`	wordsplit	BM25	206.13s
`fbComp`	`dbpComp`	lower	intersect	9.47s
`dbpComp`	`secComp`	wordsplit	BM25	243.54s
`fbDrug`	`dbankDrug`	exact	maxinc	2.49s
`fbDrug`	`dbpDrug`	wordsplit	BM25	48.56s
`dbpDrug`	`dbankDrug`	wordsplit	BM25	47.60s
`fbMovie`	`imdbMovie`	exact,lower	intersect	1.29s
`fbMovie`	`dbpMovie`	exact	intersect	5.09s
`dbpMovie`	`imdbMovie`	split	BM25	68.47s

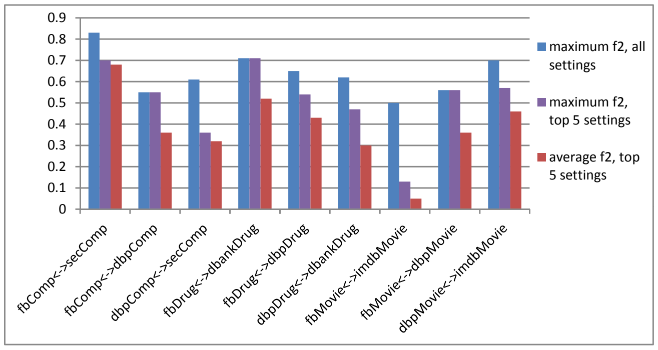


Figure 6: Accuracy of top 5 settings ranked by average match score

5.5 Finding the Right Settings

In addition to the experiments described above, we have also tested the ability of our framework to find the right analyzer, similarity function, and parameters, without having a ground truth, and solely based on the average score of the linkage points returned by the SMaSh-X algorithm. To verify this, we rank the settings (analyzer, similarity function, and parameters) by the average score of the top 5 linkage points returned by the SMaSh-X algorithm. Figure 6 shows the maximum and average F_2 score of the top 5 settings in this ranking for each of the scenarios, along with the maximum possible F_2 score in all settings. The figure shows that in three of the scenarios, the best setting is in fact among the top 5 settings, and except for two cases, the top 5 settings have F_2 score comparable to the highest F_2 score. In addition to this, we have also experimented with non-overlapping data sets and in most cases, very few results are returned by the SMaSh-X algorithm, indicating that the data sets cannot be linked.

5.6 Record Linkage Results

To show how the discovered linkage points can improve the overall record linkage process, we discuss one of the scenarios in more detail, and compare the results obtained using the baseline algorithm and one of the best performing settings of the SMaSh-X algorithm. Table 8 shows the top 10 linkage points in the Freebase-SEC Company scenario returned by the baseline algorithm, i.e., the results of finding the intersection of the set of value strings transformed into lowercase. Table 9 shows the top 10 linkage points returned by the SMaSh-X algorithm for the same scenario, using wordsplit analyzer and BM25 similarity function and the default parameters shown in Table 3.

Although the focus of this paper is not on the final record linkage algorithm, and there are many ways that the discovered linkage points can be used to establish links between the records, here we show how the discovered linkage points can be used to enhance the results of two simple linkage algorithms. The first algorithm matches two records if the values of the attributes in the linkage point match using the same matching function used to discover the linkage points. If the matching function returns a similarity score, only the records with the highest score are matched. This means that for SMaSh-S using lower analyzer, this algorithm links records that share the same (lowercase) attribute value, and for SMaSh-X using BM25 similarity, only records with attribute values having the highest similarity score are matched. The number of records that match using this algorithm are shown in column match# in Tables 8 and 9. The second algorithm uses the constraint that each record can be linked with at most one record in the other data set due to the fact that the two data sets are clean. The algorithm simply removes any one-to-many matches from the results of the first algorithm. This means that we link two records only if the matching algorithm is able to find a one-to-one relationship for the two records. The results are shown in the link# column in Tables 8 and 9. We have manually verified a subset of the linked records and estimate that the precision of this algorithm in linking records that refer to the same real-world entity is above 90% when the linkage point is accurate (those marked with $*$ in Table 8 and all the linkage points in Table 9).

The results in Table 8 show that 7 out of the top 10 linkage points returned by the baseline algorithm are accidental matches, mostly as a result of numerical values. Note that simply removing the attributes that have numerical values in advance is not possible in this case since for example the `key` attribute in freebase contains both numerical values that result in false matches, and string and numerical values that result in correct links. Also the `ticker_symbol` and `stockSymbol` attributes returned by baseline result in many inaccurate (one-to-many) links, which is a result of companies in different countries sharing the same stock symbol, and many companies having values such as `None` or `N/A` as their stock symbol. Also note that the `name` attributes link only 271 out of the 1,981 records in SEC.

Table 9 shows that the wordsplit analyzer and BM25 similarity are more effective in linking records in the two data sets, and all the discovered linkage points by the SMaSh-X algorithm are accurate (i.e., result in accurate links). It is worth mentioning that even the first linkage algorithm results in a reasonable number of links in this case, and that many one-to-many matches link companies that are not the same, but are highly related (e.g., `Pixar` and `Walt Disney` that are matched due to sharing board members). Overall, the union of all the one-to-one (and highly accurate) links established is 1,791 (out of maximum 1,981 possible links) which is 233 links (15%) more than what could be found using the accurate linkage points returned by the baseline algorithm. Note that in some other scenarios, the baseline approach returns no accurate linkage points. For example, in the DBpedia-SEC Company scenario, DBpedia does not contain the key `cik` attribute or a large number of stock symbols, whereas many numerical or date attributes exist in both data sets. Also, it is important to note that in all these real-world scenarios, there is no single *magical* linkage point that can effectively link a large number of records. Therefore, a combination of several linkage points must be used to derive the largest number and highest quality of links between the data sets.

6. DISCUSSION

In light of the results presented in Section 5, here we review the assumptions made in the design of our proposed framework, and how it fits in the overall record linkage process. One of the key assumptions we make in this work is that the input data sources are sets of records that describe a set of entities of the same type (e.g., each record in data set `fbComp` describes one entity of type *company*), although the (nested) records can contain descriptions of entities of other types (e.g., a company record can contain information about its *key people*, *location*, *subsidiary companies*, etc.). This requires the input data to either: 1) include a notion of entity type so that we can query only for one type of entity; this is the case for Freebase and DBpedia, and 2) describe only one entity type that is known to the user; this is the case for data sets like IMDb or DrugBank. In general, the input data sets may contain no information about the entity types, or the correspondence between the types between two data sets may not be clear. For example, if we consider all Freebase and all DBpedia as input, we first need to

Table 8: Top 10 linkage points returned by baseline algorithm in Freebase-SEC Company scenario

attr1	attr2	score	match#	link#
`freebase -> company -> key`	`sec -> company -> transactions -> shares`	4208	29,520	2
`freebase -> company -> key`	`sec -> company -> holdings -> shares`	3570	9,642	39
`freebase -> company -> ticker_symbol`	`sec -> company -> stockSymbol`	1421	6,392	1,370 $*$
`freebase -> company -> key`	`sec -> company -> cik`	1,381	1,381	1,375 $*$
`freebase -> company -> founded`	`sec -> company -> transactions -> shares`	317	1,116,105	0
`··· -> headquarters -> postal_code`	`sec -> company -> holdings -> shares`	291	4,141	21
`··· -> number_of_employees -> number`	`sec -> company -> transactions -> shares`	290	45,324	2
`··· -> headquarters -> postal_code`	`sec -> company -> transactions -> shares`	286	10,219	7
`freebase -> company -> name`	`sec -> company -> name`	271	271	271 $*$
`··· -> number_of_employees -> number`	`sec -> company -> holdings -> shares`	262	12,194	1
Total number of records accurately linked using the above linkage points (those marked with $*$):				1,558

Table 9: Top 10 linkage points returned by SMaSh-X algorithm in Freebase-SEC Company scenario

attr1	attr2	score	match#	link#
`freebase -> company -> id`	`sec -> company -> name`	20.21	667	649
`freebase -> company -> name`	`sec -> company -> name`	19.88	1,570	990
`freebase -> company -> key`	`sec -> company -> name`	18.77	1,575	986
`freebase -> company -> major_shareholders -> name`	`sec -> company -> officers -> name`	18.45	151	113
`freebase -> company -> major_shareholders -> id`	`sec -> company -> officers -> name`	17.66	124	100
`freebase -> company -> key`	`sec -> company -> cik`	15.21	1,381	1,375
`freebase -> company -> major_shareholders -> name`	`sec -> company -> directors -> name`	14.62	295	150
`freebase -> company -> major_shareholders -> id`	`sec -> company -> directors -> name`	13.86	229	139
`··· -> /business/employer/employees -> person -> name`	`sec -> company -> officers -> name`	13.63	1,766	679
`··· -> /business/employer/employees -> person -> id`	`sec -> company -> officers -> name`	13.52	1,055	917
Total number of records linked using the above linkage points:				1,791

identify that entities of type *organization* in Freebase are related to entities of type *company* in DBpedia ontology. For our framework to work in such cases, an extra step of identification of entity types and their relationship is required. For the case of RDF data sets, this problem is a specific type of ontology matching problem where the goal is matching (RDF) classes. In our recent work [7] we study this problem and propose an efficient type matching method based on locality sensitive hashing. Combining the two steps and providing a large-scale end-to-end matching of data sets of mixed types is an interesting direction for future work.

Once the linkage points are identified, there are several ways to perform or improve the record linkage using the output of our framework. In Section 5.6, we showed the results of two linkage methods that both follow the basic approach of using the top-k linkage points returned, take each individual linkage point to perform the linkage, and then find the union of the linkage sets. However, one of the main motivations of this work is to provide a framework for the semi-automatic construction of ER rules that start from the linkage points returned by the algorithms developed in this paper. Declarative systems such as Dedupalog [1] or HIL [11] can use our linkage points, which already have good linkage properties on an individual basis, as building blocks that are combined in various logical expressions and multiple rules to achieve even higher precision and recall. For example, a rule could match two entities by using "name" AND "phone" OR by using "name" AND "address", where "name", "phone", and "address" would be linkage points pre-discovered using our framework. In effect, our linkage points form the basis for the subsequent development of the various rules for entity resolution. We are currently investigating the applicability of our framework in the context of such declarative languages for entity resolution. Note that not only better linkage rules can be developed using a ranked list of linkage points, it is also possible to improve the efficiency of the linkage process. For example, if a setting of the SMaSh-X algorithm that uses the exact analyzer results in one linkage point with a high coverage and strength, and several settings that use q-gram analyzer result in multiple good-quality linkage points, it is more efficient to execute the exact matching of the values first, and then perform the more expensive q-gram based matching on the remaining records.

In Section 5.5, we showed how our framework can be used to find the right settings without having a ground truth. The result we showed was based on running the algorithms with several different analyzers, similarity functions, and parameters, and ranking the settings based on the average score of the top linkage points returned. It is important to note that such extensive running of the algorithms using different settings is not required in practice. First, the quality of the linkage points returned by one setting can often be used to estimate the performance of another setting. For example, if a 3-gram analyzer works very well (or very poorly), we can expect that a 4-gram analyzer cannot perform significantly better and so there is no need to run the algorithm with this analyzer. Moreover, the task scheduler along with the system's API and Web interface play an important role in determining the right settings. The user can verify the partial results returned by each setting while the algorithms are still running, and stop running tasks or start new ones based on the previous results. Even if the user decides to run tens of settings (as done for the experiments in Section 5.5) the task scheduler will handle running of the algorithms using a simple configuration file and only a few calls to the API (or clicks on the Web interface). Our experience in finding optimal settings in the scenarios described in this paper and other real-world scenarios has been very promising. Nonetheless, an interesting direction for future work is adopting state-of-the-art automatic algorithm configuration techniques such as ParamILS [12] in our framework.

7. RELATED WORK

As mentioned earlier, our work is closely related to the very active research areas of ontology matching, (database) schema matching, and duplicate detection. Here, we only provide a brief overview of some recent existing work related to ours, and refer the readers to recent survey articles and books for a complete overview of the work in these areas [5, 8, 16, 18].

In comparison to schema matching, our work can be seen as an all-to-all instance-based matching that has been recognized as extremely useful but prohibitively expensive operation in the literature [18, page 343]. Based on the classification of the ontology matching techniques by Euzenat and Shvaiko [8, page 105], our work can be seen as fast, approximate *matching-based extension* comparison techniques, although our final goal in this paper is data interlinking and not providing a full alignment. Two closely related instance-based matching approaches are the work of Warren and Tompa [21] and iMAP [6]. They also use a search strategy to find schema correspondences, and take advantage of the information gained from the overlapping instances. However, we use a novel and highly efficient search strategy that treats the matching function as a black-box and uses specific measures to reduce search space and improve search results. This allows the careful study and evaluation of the search algorithms we have performed in this work, and application on very large heterogeneous schema-free (Web) data sources as opposed to relational data with a limited number of attributes.

Our work can also be seen as a first step towards automatic discovery of linkage rules (or linkage specifications). To the best of our knowledge, this problem has only recently been studied for Web data, and as a part of SILK [13] and LIMES [17] RDF link discovery frameworks. These approaches use learning algorithms and rely on a number of manually-labeled link candidates. The SILK framework [13] uses a genetic programming paradigm to learn linkage rules from a set of reference links. LIMES's RAVEN algorithm [17] has the advantage of performing matching of classes and therefore does not require matching of entities of the same type. On the other hand, it also requires a number of manually-labeled link candidates to perform linking, and a user-specified threshold factor. Our approach is complimentary to such learning approaches as it can provide a number of candidate linkage points (as opposed to a single best-performing rule that uses a fixed set of attributes), and can make use of a larger set of instance values. This is also the reason we have not used the same data sets and linkage rules from previous work as ground truth to evaluate our framework. An important issue here is that even if source and target data describe the same entity types, different portions of instances may require different linkage rules. For example, a small portion of data may contain identifiers that can be used to effectively discover links (i.e., are strong linkage points). Our experiments over real data confirm that our framework is capable of finding such linkage points.

8. CONCLUSION

In this paper, we presented a framework for discovering linkage points over large semistructured Web data. Our framework includes a library of lexical analyzers, a library of similarity functions, a set of novel search algorithms along with effective filtering strategies. We experimentally evaluated our framework in nine scenarios involving real Web data sources in three domains. The data sets used in our experiments along with the results are available [30]. We are currently exploring a number of interesting directions for future work. We are planning to further extend our similarity function library to include functions that take into account semantic similarity of the values in addition to syntactic and lexical matching. Our goal is to extend our system to publish the linkage points we discover using existing standards such as the recently proposed R2R mapping language [2]. In addition, we are planning to use the SMaSh algorithms to facilitate online analytics of enterprise data using external Web repositories as a part of the Helix project [9].

9. REFERENCES

[1] A. Arasu, C. Ré, and D. Suciu. Large-Scale Deduplication with Constraints Using Dedupalog. In *Proc. of the IEEE Int'l Conf. on Data Eng.*, pages 952–963, 2009.

[2] C. Bizer and A. Schultz. The R2R Framework: Publishing and Discovering Mappings on the Web. In *ISWC Workshop on Consuming Linked Data (COLD)*, 2010.

[3] J. Bleiholder and F. Naumann. Data Fusion. *ACM Computing Surveys*, 41(1), 2008.

[4] D. Burdick, M. A. Hernández, H. Ho, G. Koutrika, R. Krishnamurthy, L. Popa, I. Stanoi, S. Vaithyanathan, and S. R. Das. Extracting, Linking and Integrating Data from Public Sources: A Financial Case Study. *IEEE Data Engineering Bulletin*, 34(3):60–67, 2011.

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

[6] R. Dhamanka, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 383–394, 2004.

[7] 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 *Proc. of the Int'l Semantic Web Conference (ISWC)*, pages 49–64, 2012.

[8] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer-Verlag, 2007. http://book.ontologymatching.org/.

[9] O. Hassanzadeh, S. Duan, A. Fokoue, A. Kementsietsidis, K. Srinivas, and M. J. Ward. Helix: Online Enterprise Data Analytics. In *Int'l World Wide Web Conference (WWW)*, pages 225–228, 2011.

[10] O. Hassanzadeh, R. Xin, R. J. Miller, A. Kementsietsidis, L. Lim, and M. Wang. Linkage Query Writer. *Proceedings of the VLDB Endowment (PVLDB)*, 2(2):1590–1593, 2009.

[11] M. A. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, and R. Wisnesky. HIL: A High-Level Scripting Language for Entity Integration. In *Advances in Database Technology - Int'l Conf. on Extending Database Technology (EDBT)*, pages 549–560, 2013.

[12] F. Hutter, H. H. Hoos, and T. Stützle. Automatic Algorithm Configuration Based on Local Search. In *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1152–1157, 2007.

[13] R. Isele and C. Bizer. Learning Expressive Linkage Rules using Genetic Programming. *Proceedings of the VLDB Endowment (PVLDB)*, 5(11):1638–1649, 2012.

[14] J. Kang and J. F. Naughton. On Schema Matching with Opaque Column Names and Data Values. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 205–216, 2003.

[15] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[16] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[17] A.-C. Ngonga Ngomo, J. Lehmann, Sören Auer, and K. Höffner. RAVEN - Active Learning of Link Specifications. In *ISWC Workshop on Ontology Matching*, 2011.

[18] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The Int'l Journal on Very Large Data Bases*, 10(4):334–350, 2001.

[19] S. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009.

[20] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[21] R. H. Warren and F. Wm. Tompa. Multi-column Substring Matching for Database Schema Translation. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 331–342, 2006.

[22] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On Multi-Column Foreign Key Discovery. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1):805–814, 2010.

[23] DBpedia. `http://dbpedia.org/`. [Online; accessed 12-12-2012].

[24] DrugBank. `http://drugbank.ca/`. [Online; accessed 12-12-2012].

[25] IMDb. `http://www.imdb.com/`. [Online; accessed 12-12-2012].

[26] Redis. `http://redis.io/`. [Online; accessed 12-12-2012].

[27] Xapian. `http://xapian.org/`. [Online; accessed 12-12-2012].

[28] Freebase. `http://freebase.com/`. [Online; accessed 12-12-2012].

[29] The U.S. Securities & Exchange Commission's Public Information Server. `http://ftp.sec.gov/`. [Online; accessed 12-12-2012].

[30] Project Data Sets & Results. `http://purl.org/linkdiscovery/touch/data`.