Normalization of Duplicate Records from Multiple Sources

Yongquan Dong, Eduard C. Dragut, *Member, IEEE,* and Weiyi Meng, *Senior Member, IEEE*

Abstract—Data consolidation is a challenging issue in data integration. The usefulness of data increases when it is linked and fused with other data from numerous (Web) sources. The promise of Big Data hinges upon addressing several big data integration challenges, such as record linkage at scale, real-time data fusion, and integrating Deep Web. Although much work has been conducted on these problems, there is limited work on creating a uniform, standard record from a group of records corresponding to the same real-world entity. We refer to this task as *record normalization*. Such a record representation, coined *normalized record*, is important for both front-end and back-end applications. In this paper, we formalize the record normalization problem, present in-depth analysis of normalization granularity levels (e.g., *record*, *field*, and *value-component*) and of normalization forms (e.g., *typical* versus *complete*). We propose a comprehensive framework for computing the normalized record. The proposed framework includes a suit of record normalization methods, from naive ones, which use only the information gathered from records themselves, to complex strategies, which globally mine a group of duplicate records before selecting a value for an attribute of a normalized record. We conducted extensive empirical studies with all the proposed methods. We indicate the weaknesses and strengths of each of them and recommend the ones to be used in practice.

Index Terms—Record normalization, data quality, data fusion, web data integration, deep web

1 Introduction

The Web has evolved into a data-rich repository containing a large amount of structured content spread across millions of sources. The usefulness of Web data increases exponentially (e.g., building knowledge bases, Web-scale data analytics) when it is linked across numerous sources. Structured data on the Web resides in Web databases [1] and Web tables [2]. Web data integration is an important component of many applications collecting data from Web databases, such as Web data warehousing (e.g., Google and Bing Shopping; Google Scholar), data aggregation (e.g., product and service reviews), and metasearching [3].

Integration systems at Web scale need to automatically match records from different sources that refer to the same real-world entity [4], [5], [6], find the true matching records among them and turn this set of records into a standard record for the consumption of users or other applications. There is a large body of work on the *record matching problem* [7] and the *truth discovery problem* [8]. The record matching problem is also referred to as duplicate record detection [9], record linkage [10], object identification [11], entity resolution [12], or deduplication [13] and the truth discovery problem is also called as truth finding [14] or fact finding [15] - a key problem in data fusion [16], [17]. In this paper, we assume that the tasks of record matching and truth discovery have been performed and that the groups of true matching records have thus been identified. Our goal is to generate a uniform, standard record for each group of true matching records for end-user consumption. We call the generated record the *normalized record*. We call the problem of computing the normalized record for a group of matching records the *record normalization problem* (RNP), and it is the focus of this work. RNP is another specific interesting problem in data fusion.

Record normalization is important in many application domains. For example, in the research publication domain, although the integrator website, such as Citeseer or Google Scholar, contains records gathered from a variety of sources using automated extraction techniques, it must display a normalized record to users. Otherwise, it is unclear what can be presented to users: (i) present the entire group of matching records or (ii) simply present some random record from the group, to just name a couple of ad-hoc approaches. Either of these choices can lead to a frustrating experience for a user, because in (i) the user needs to sort/browse through a potentially large number of duplicate records, and in (ii) we run the risk of presenting a record with missing or incorrect pieces of data.

Record normalization is a challenging problem because different Web sources may represent the attribute values of an entity in different ways or even provide conflicting data. Conflicting data may occur because of incomplete data, different data representations, missing attribute values, and even erroneous data. For example, Table 1 contains four records corresponding to the same entity (publication). They are extracted from different websites. Record R_{norm} is constructed by hand for illustration purposes. One notices that the same publication has different representations in different websites. For instance, the field `author` uses the

- *Y. Dong is with the School of Computer Science and Technology, Jiangsu Normal University, Xuzhou, Jiangsu, China 221116. E-mail: tomdyq@163.com*
- *E.C. Dragut is with the Computer and Information Sciences, Temple University, Philadelphia, PA 19022. E-mail:edragut@temple.edu*
- *W. Meng is with the Computer Science Department, Binghamton University, Binghamton, NY 13902. E-mail: meng@cs.binghamton.edu*

Manuscript received October 1, 2017.

TABLE 1
Four records for the same publication: R_a, R_b, R_c, and R_d are extracted from different websites and R_{norm} is constructed manually.

Fields	author	title	venue	date	pages
$\mathbf{R_a}$	Halevy, A.; Rajaraman A.; Ordille, J.	Data integration: the teenage years	in proc 32nd int conf on Very large data bases	2006	
$\mathbf{R_b}$	A. Halevy, A. Rajaraman, J. Ordille	Data integration: the teenage years	in VLDB	2006	9-16
$\mathbf{R_c}$	A. Halevy, A. Rajaraman, J. Ordille	Data integration: the teenage years	in proc 32nd conf on Very large data bases	2006	pp.9-16
$\mathbf{R_d}$	A. Halevy, A. Rajaraman, J. Ordille	Data integration: the teenage years		2006	9-16
$\mathbf{R_{norm}}$	Alon Halevy, Anand Rajaraman, Joann Ordille	Data integration: the teenage years	in proceedings of the 32nd international conference on Very large data bases	2006	9-16
$\mathbf{R_{field}}$	A. Halevy, A. Rajaraman, J. Ordille	Data integration: the teenage years	in proc 32nd int conf on Very large data bases	2006	pp.9-16

format "last-name, first-name-initial" in the record R_a, but the values of the same field in the records R_b, R_c, and R_d use the format "first-name-initial. last-name". One can also observe that the value of the field `pages` is absent in R_a. The field `venue` has incomplete values in three of the four records and has no value in R_d; it contains the abbreviations "proc", "int", "conf" to represent "proceedings", "international" and "conference", respectively, in the records R_a and R_c; it contains the acronym "VLDB" to represent "Very Large Data Bases" while missing "proceedings of the 32nd international conference on" in R_b. Some values of the attributes of R_{norm} cannot be acquired directly from the given set of matching records, such as the first names of the authors. They could be obtained by mining external sources, such as a search engine. In this paper, we focus on the *best-effort record normalization*: we compute R_{norm} from the set of matching records and do not explore external sources. Furthermore, this paper only focuses on the normalization of text data, and we will leave the normalization of data involving numeric and more complex values as future work.

Brief Overview of the Proposed Solution

We identify three levels of normalization granularity: *record*, *field*, and *value-component*.

Record level assumes that the values of the fields within a record are governed by some hidden criterion and that together create a cohesive unit that is user-friendly. As a consequence, this normalization favors building the normalized record from entire records among the set of matching records rather than piecing it together from field values of different records. Thus, any of the matching records (ideally, that has no missing values) can be the normalized record. Using our running example in Table 1, the record R_c is a possible choice for the normalized record with this level of normalization granularity.

Field level assumes that record level is often inadequate in practice because records contain fields with incomplete values. Recall that these records are the products of automatic data extraction tools, which are not perfect and thus may produce errors [18]. This normalization level ignores the cohesion factor in the record normalization level and assumes that a user is better served when each field of the normalized record has as easy to understand a value as possible, selected from among the values in the set of matching records. It treats each field of the normalized record independently, finds a normalized value (according to some criterion) per field, and creates the normalized record by stitching together the normalized values of the fields. The normalized record may not resemble any of the matching records, but it will convey the same information as any of them, in a user-friendlier form than any of the individual records. For example, consider the field `venue` of R_{field}. We may take (according to a number of criteria that we will describe in later sections) the value "in proc 32nd int conf on Very large data bases" from record R_a (Table 1) as its normalized value.

Value-component level takes the field level normalization a step "deeper." It assumes that in general the value of a field may comprise of multiple pieces some of which may not be easy to grasp by an ordinary user. For example, a field (such as `venue`) may contain arcane acronyms illegible to an ordinary user. A normalization solution in accordance with this level will yield a value for a field with the property that the individual components of the value are themselves normalized. The resulted (normalized) value may not physically exist in any of the matching records. For example, the values of R_a, R_b, and R_c for the field `venue` contain acronyms, incomplete, and unexpanded terms. We can synthesize a normalized value for this field by mining the set of records and make the following inferences:

- "proc", "int", "conf" are the abbreviations of "proceedings", "international" and "conference", respectively, and
- the collocation "in proceedings of the" appears frequently as a whole unit.

Thus, we can create a normalized value for `venue`, at the value-component level, as follows.

1) We take the value suggested previously by the field-level for `venue` and replace the abbreviations in it with the complete words and change it into "in proceedings 32nd international conference on Very large data bases".
2) We find that "in proceedings" is the part of the collocation "in proceedings of the".
3) We use the collocation to replace "in proceedings".
4) Finally, we get the normalized value of `venue`, "in proceedings of the 32nd international conference on Very large data bases".

A quick visual inspection of the records $R_a - R_d$ shows that this value, although desirable, is not present in any of these records. After each field gets its normalized value

according to the value-component level, we piece them together to create the normalized record.

Naive solutions to RNP are often inadequate. For example, one simple solution for the field-level normalization is to return the most common string of each field as its normalized field value. However, this strategy is inadequate in the presence of records with missing values. In our running example, this approach will produce the value "in proc 32nd int conf on Very large data bases" for the field `venue`, but the value "in proceedings of the 32nd international conference on Very large data bases" is clearly much better when complete citation information is desirable. Providing non-naive strategies to the three normalization levels is a challenging task. For example, a key challenge in providing a solution according to value-component level is that a value-component may comprise multiple adjacent pieces and the value of a field may contain components with uneven lengths (e.g., "in proceedings of the" and "conf" are value components in `venue`). They need to be discovered and normalized, computationally.

Contributions

In this paper we aim to develop a framework for constructing normalized records systematically. This paper has the following contributions:

- We propose three levels of granularities for record normalization along with methods to construct normalized records according to them.
- We propose a comprehensive framework for systematic construction of normalized records. Our framework is flexible and allows new strategies to be added with ease. To our knowledge, this is the first piece of work to propose such a detailed framework.
- We propose and compare a range of normalization strategies, from frequency, length, centroid and feature-based to more complex ones that utilize result merging models from information retrieval, such as (weighted) Borda.
- We introduce a number of heuristic rules to mine desirable value components from a field. We use them to construct the normalized value for the field.
- We perform empirical studies on publication records. The experimental results show that the proposed weighted-Borda-based approach significantly outperforms the baseline approaches.

The rest of the paper is organized as follows. Section 2 defines the problem. Section 3 introduces the granularity levels for record normalization. Section 4 presents the overall framework and the normalization techniques. Section 5 reports the experimental results. Section 6 gives a brief overview of the related work. Section 7 concludes the paper and discusses several open research issues.

2 Problem Definition

Let E be a set of real-word entities relevant for the application domain at hand, say scientific publications. Denote by $R^e = \{r_1, r_2, \ldots, r_{ne}\}$ the set of matching records that refer to an entity $e \in E$, where ne is the number of the matching records for the entity e, $|R^e| = ne$. The records may be collected from Web databases (e.g., ACM Digital Library) or from ad-hoc publication lists (e.g., author home pages). The entity e has a set of fields (attributes), $FS = \{f_1, f_2, \ldots, f_{|FS|}\}$, where $|FS|$ is the number of the fields of the entity e. We use the notation $r_i[f_j]$ to refer to the value of the field f_j in the record r_i. We assume the NULL value for each field without a value.

Record Normalization Problem (RNP): Create a normalized record nr_e for each entity $e \in E$ from the set of matching records R^e that summarizes the information about e as accurately as possible.

Currently, there is not a widely accepted standard for record normalization, but there are a few prerequisites of a good normalized record:

(1) *Error-free*: A normalized record should avoid errors, such as misspellings or incorrect field values, as much as possible.

(2) *Comprehensive*: A normalized record should contain a value for each field whenever possible.

(3) *Representative*: A normalized record should reflect the commonality among the matched records.

3 Normalization Granularities and Forms

In this section, we first present three levels of record normalization. Then we give two forms of normalization.

3.1 Levels of Record Normalization

We propose three levels of normalization: record, field, and value-component. Note that regardless of the chosen level of normalization, the goal is to provide users with some form of normalized record that is the easiest to grasp by an ordinary user.

3.1.1 Record-level Normalization

The record-level normalization assumes that each record $r_i \in R^e$ is a cohesive unit, in the sense that taken together the values $r_i[f_j]$ of the fields f_j in r_i give a coherent depiction of entity e. The assumption, while intuitively appealing and allows to build the theoretical underpins for constructing normalized records, needs to be taken with a grain of salt in practice. R^e contains a mixture of candidate normalized records and records with incomplete or arcane representations of e, which may be difficult to understand by ordinary users. The challenge is to select a record $r_i \in R^e$ that is most likely to be a reasonable candidate. The selection can be performed according to several criteria (described in Section 4.1). One elementary criterion is to demand that the selected record must have a value for each field. Note that R_c in Table 1 meets the constraints of this strategy.

3.1.2 Field-level Normalization

Field-level normalization selects a normalized value for each field f_i independently and concatenates the selected values of all fields into a normalized record. The normalized value for the field f_i is one of the values that appear among the records in R^e in the field f_i and it is selected according to some criteria (e.g., more descriptive). The normalized record formed in this way may consist of field values from different records. For example, R_{field} in Table 1 is the normalized

record constructed out of the field values of R_a - R_d. The values of R_{field} in the fields `venue` and `pages` are taken from R_a and R_c, respectively, because they are the most descriptive. The record obtained by concatenating these field values does not exist among the matching records. In general, the normalized record may not correspond to any of the original set of matching records.

3.1.3 Value-component-level Normalization

Value-component level is at an even finer granularity than the field-level. It seeks to create a normalized field value v^i_{norm} for a field f_i that is as expressive as possible (to minimize ambiguity) but still semantically equivalent to any of the (correct) values $r_j[f_i]$, $r_j \in R^e$. It builds on the assumption that $r_i[f_j]$ is a concatenation of components $c^{i,j}_1 c^{i,j}_2 ... c^{i,j}_k$. For example, the components of `venue` in R_c are: "in proc," "32nd," "int," "conf," "on," and "Very large data bases." We note that some of the components $c^{i,j}_t$ are incomplete (e.g., "in proc"). Incompleteness can take several forms. For instance, $c^{i,j}_t$ may be a half-finished collocation, such as "in proc," or an abbreviation, such as "conf." Our goal here is two-fold: (1) Detect the incomplete components $c^{i,j}_t$ of a field value and (2) for each incomplete $c^{i,j}_t$ find an (equivalent) replacement $d^{i,j}_t$ that addresses its incompleteness. In our running example, if $c^{i,j}_t$ = "conf" then $d^{i,j}_t$ = "conference." In this work, we assume that $d^{i,j}_t$ is present among the records in R^e. We leave the task of extracting $d^{i,j}_t$ from external sources for future work. Under this (finer level) normalization goal, not only we may generate a normalized record that does not appear in R^e, but the field values of the normalized record themselves may not appear in R^e.

3.2 Normalization Forms

We present two forms of normalization for a normalized record: *typical* and *complete*.

3.2.1 Typical Normalization

The purpose of typical normalization is to produce a normalized record that resemblances many of the matching records without modifying any of the field values. One way to define it is by frequency of occurrence. With this definition, the record-level normalization will yield a record representation that appears most often among the set of matching records for an entity. The field-level normalization will select the most frequent value for each field in the normalized record. Other strategies are clearly conceivable to perform typical normalization and we present additional alternatives in Section 4. The value-component level normalization inherently does not produce *typical* normalized records because it may create *new* values for some of the fields of the normalized records.

3.2.2 Complete Normalization

Complete normalization seeks to produce the normalized record with the property that the value of each of its fields is both complete (not missing component) and self-explanatory. For example, there are several different representations of an author's name, such as full name versus first name initial and last name. One would consider the former to be a better, less ambiguous representation of an author's name than the latter. Likewise, a fully spelled out conference name or journal name is better than its abbreviated counterpart. A record in this form of normalization is unique modulo certain set of transformations, such as permutation (e.g., "the 32nd international conference on Very large data bases, in proceedings of") or replacement with equally unambiguous (e.g., "in proceedings of the thirty second international conference on Very large data bases") of value components. This form of normalization is difficult to achieve in practice. Instead, we strive to produce a version of the normalized record as complete and self-explanatory as possible given the data at hand. Only the value-component-level strategy can achieve this form of normalization. The reason is that normalization at the record-level and field-level are inherently confined to work with monolithic field values (not value components) from the matching records, which are often incomplete.

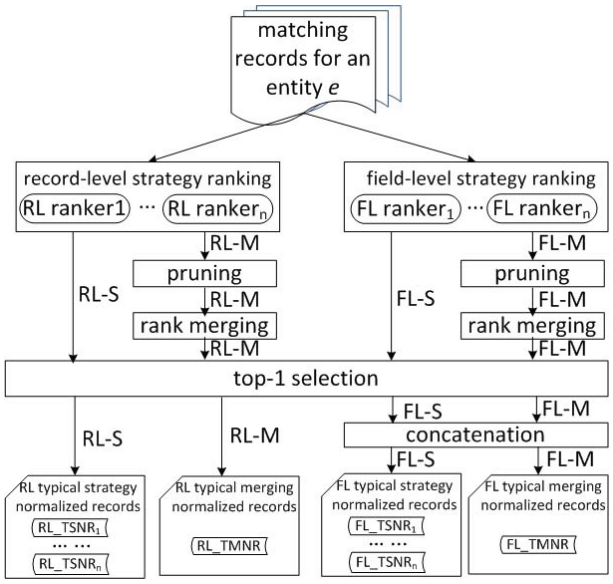


Fig. 1. The typical normalization framework.

4 Our Approach

In this section, we first present our overall framework. Then, we give the details of our solutions.

4.1 Solution Framework

We follow different steps for the two normalization forms. Fig. 1 shows the steps of the typical normalization framework and Fig. 2 shows those of the complete normalization framework.

In both frameworks, the input is the set of matching records R^e for an entity e. Different normalization strategies may be employed at each step in the normalization framework. Different choices will yield different normalized records for the same set of matching records. The normalized records are represented by parallelograms in Fig.1 and Fig.2. At every granularity level, we perform two categories of approaches: *single-strategy* and *multi-strategy* approaches. In Fig. 1 and Fig. 2, the string suffix "-S" on

the arrows denotes a single-strategy approach and "-M" denotes a multi-strategy approach; "RL" stands for "record-level", "FL" stands for "field-level" and "VCL" stands for "value-component-level".

4.1.1 Typical Normalization Framework

The typical normalization framework has two paths (Fig. 1): record-level and field-level. The former works with whole records from R^e. It includes a number of record-level rankers (RL rankers) to rank the records in R^e according to their fitness to represent the normalized record for entity e. In the single-strategy approach, each ranker recommends the top-1 candidate in its ranked list as the normalized record. In Fig. 1, RL_TSNR_i denotes the normalized record recommended by the i^{th} ranker. If we instead use the multi-strategy approach, then we employ rank merging methodologies [3] to select the final normalized record. In the multi-strategy approach each ranker acts as a voter and the records in R^e are the candidates (for the normalized record). Each ranker ranks the records in descending order of preference. After pruning out the records which have small probabilities to become the normalized record, only the top-k records are kept at each ranker as possible candidates for the normalized record. The ranked lists of records produced independently by rankers are merged into a global ranked list. The top-1 candidate record of the global list becomes the normalized record.

The typical normalization with field-level granularity works with whole field values. It includes a range of field-level rankers (FL rankers) to rank the field values of a field based on their fitness to serve as the normalized value for that field. The single-strategy approach uses one value ranker per field. The top candidates for each field are concatenated to construct the normalized record. The multi-strategy approach employs multiple value rankers per field f_j; it merges the top-k ranked lists of values produced by the various rankers for f_j and selects the top value as the normalized value for f_j. The final normalized record is constructed by taking the normalized value of each field f_j.

4.1.2 Complete Normalization Framework

The complete normalization form works at the value-component granularity level. It first performs a pre-processing step to consolidate each field format into a single format across all records in R^e. For example, the field (author) `name` is consolidated into "last-name first-name". Then it uses field-level rankers to rank the values of every field. Next, it prunes out some of the values that are unlikely to become the normalized value for that field. The pruning is discussed in Section 5.3.3. It divides the values of a field into *components* and mines them to determine a more consistent and legible (by ordinary users) value for the field. The single- and multi-strategy approaches are applied here similarly as described in Section 4.1.1.

In the following sections, we give the details of our key techniques: (1) ranking-based strategies, (2) value component mining, and (3) ranked list merging.

4.2 Ranking-based Strategies

We utilize four ranking strategies: *frequency*, *length*, *centroid*, and *feature-based*. We use them to construct several rankers

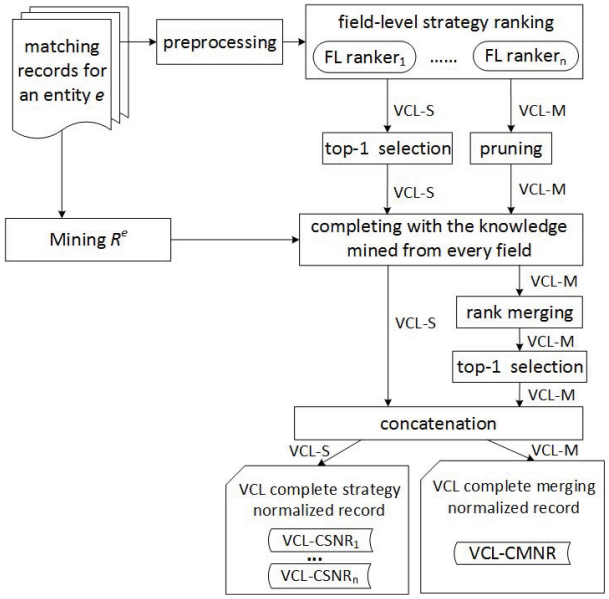


Fig. 2. The complete normalization framework.

at record and field levels. To give a uniform presentation, we refer to records and their fields as units in this section. Let U be a bag of units for the same entity e. (It is a bag because the same value or the same record may appear multiple times.) U has p distinct units denoted by $\overline{U} = \{u_1, \ldots, u_p\}$. If a ranker γ ranks a unit u higher than another unit v then we interpret this as saying that u is more appropriate as a normalized unit than v, according to γ.

4.2.1 Frequency Ranker

This ranker is defined as the ordered list of distinct units

$$FR(U) = [u_1, \ldots, u_p], \tag{1}$$

where u_i appears more frequently than u_j in U, for $i < j$.

4.2.2 Length Ranker

Length ranker is defined as the ordered list of distinct units

$$LR(U) = [u_1, \ldots, u_p], \tag{2}$$

where the character length of u_i is larger than that of u_j, $1 \leq i < j \leq p$.

4.2.3 Centroid Ranker

Let *SM* be a similarity measure between units. We define the *unit centroid score* of $u \in \overline{U}$ to be

$$UCS(u) = \frac{1}{|U|^2} \sum_{v \in \overline{U}} \alpha_u \alpha_v SM(u, v) \tag{3}$$

where α_u , α_v denote the occurrence frequencies of u and v in U, respectively. The centroid ranker gives the ordered list of distinct units

$$CR(U) = [u_1, \ldots, u_p], \tag{4}$$

where $UCS(u_i) \geq UCS(u_j)$, $1 \leq i < j \leq p$.

We use three similarity measures for SM: edit-distance, bigram, and Winkler similarity.

(1) Edit-distance is the number of edit operations necessary to transform one string into another [19]. The edit-distance between strings a and b is computed as follows:

$$Sim_{ed}(a,b) = \frac{ed(a,b)}{max(|a|,|b|)} \quad (5)$$

where $|a|$ and $|b|$ denote the lengths of a and b, respectively. and $ed(a,b)$ is the edit distance between these two strings.

(2) The bigram similarity measure is based on 2-character sub-strings contained in a string. The bigram similarity measure between strings a and b is computed as follows:

$$Sim_{bigram}(a,b) = \frac{2 \times |bigram(a) \cap bigram(b)|}{|bigram(a)| + |bigram(b)|} \quad (6)$$

where $bigram(a)$ and $bigram(b)$ denote the bags of 2-grams of the strings a and b, respectively.

(3) The Winkler similarity measure is based on Jaro metric which is given by the number and order of the common characters in them [20]. The Winkler similarity measure between strings a and b is computed as follows:

$$Sim_W(a,b) = Jaro(a,b) + \frac{max(P,4)}{10}(1 - Jaro(a,b)) \quad (7)$$

where $Jaro(a,b)$ is the Jaro similarity between a and b. P is the length of the longest common prefix of a and b.

4.2.4 Feature-based Ranker

For $u \in \overline{U}$, let $\Phi(u) = \{\phi_1(u), \ldots, \phi_k(u)\}$ be a vector of *binary feature functions* $\phi : \overline{U} \rightarrow \{0,1\}$ that compute evidence indicating whether u should be selected as the normalized unit, where k denotes the number of features. For example, the value of the j^{th} feature function $\phi_j(u)$ may be 1 if unit u is ranked as the first one by the length ranker. Let $\Theta = \{\theta_1, \ldots, \theta_k\}$ be a vector of real-valued weights associated with the features.

We can compute a score for the event that u is chosen as the normalized unit by taking the dot product of the feature vector and weight vector:

$$\tau(u, \Theta) = \Phi(u) \cdot \Theta \quad (8)$$

Let the binary random variable C_u be 1 if unit u is the normalized unit of U. Given Θ and u, we can compute the probability of C_u (denoted by $pn(u)$) as:

$$pn(u) = p(C_u = 1|u, \Theta) = \frac{g(\tau(u,\Theta))}{\sum_{u \in \overline{U}} g(\tau(u,\Theta))} \quad (9)$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad (10)$$

where the score for the unit u is normalized by the scores for every other matching unit. g is the standard logistic function.

Feature-based ranker is defined as the ordered list of distinct units

$$FBR(U) = [u_1, \ldots, u_p], \quad (11)$$

where $pn(u_i) \geq pn(u_j)$, $1 \leq i < j \leq p$.

Let $TS = \{< v_1, l_1 > \ldots < v_{|TS|}, l_{|TS|} >\}$ be a training set, where v_i is the i^{th} record in the training set. and

$$l_i = \begin{cases} 1 & \text{if } v_i \text{ is the normalized unit of an entity} \\ 0 & \text{otherwise.} \end{cases}$$

We estimate Θ from the training set by minimizing the cost function $L(\Theta, TS)$ of the data:

$$L(\Theta, TS) = -\frac{1}{|TS|} \sum_{v_i \in TS} [l_i \, log(g(\tau(v_i, \Theta))) + (1 - l_i) log(1 - g(\tau(v_i, \Theta))] \quad (12)$$

We use $L2$ regularization to penalize the overall cost of L to mitigate over-fitting. We find the setting of Θ that minimizes Equation (12) using the limited-memory BFGS, a gradient ascent method with a second-order approximation [21].

The features for the feature-based rankers are as follows:

Strategy features. These features are all binary, indicating if a unit is the first, second, or third highest ranked unit according to some strategy ranker.

Text features. We compute two features that examine the properties of the strings themselves. One is the acronym feature which is true if the matching unit contains a token in a list of known acronyms (e.g., "VLDB" in our running example). Another is the abbreviation feature which is true, if the matching unit contains a token in a list of known abbreviations (e.g., "conf" for "conference"). The acronym list is obtained from the Web (e.g., www.acronymfinder.com) and the abbreviation list is mined from the existing dataset which will be given in Section 4.3.1.

4.3 Value Component Mining

We begin this section with a number of definitions to make the following description clear and consistent. Let $Val(f_j) = \{r_i[f_j] | r_i \in R^e\}$ be the collection of all values of the field f_j among the records in R^e.

Definition 4.1. The **inverse document frequency(idf)** of a term or a consecutive sequence of terms c is defined as

$$idf(c, R^e) = \frac{|R^e|}{|\{r_i | r_i \in R^e, c \in r_i[f_j]\}|} \quad (13)$$

where $|\cdot|$ denotes set cardinality (the number of records in our case). Note that when c's frequency increases, c's idf decreases.

Definition 4.2. A **collocation** is a sequence of consecutive terms in $r_i[f_j]$ with the property that its idf is less than a given threshold η_{idf}. The **length of a collocation** is the number of words (terms) it contains. **n-collocation** denotes a collocation of length n (terms).

For example, in the field `venue`, "proceedings of" is a 2-collocation, "in proceedings of" is a 3-collocation and "in proceedings of the " is a 4-collocation.

Definition 4.3. A k-collocation kc is a **subcollocation** of an n-collocation nc if kc is a substring of nc (implicitly, $k < n$).

For example, "proceedings of" is a subcollocation of "in proceedings of" which in turn is a subcollocation of "in proceedings of the".

Definition 4.4. An n-collocation c is a **template collocation** if it is not a subcollocation of any other collocation.

Note that whether or not an n-collocation is a template collocation depends on the value of the threshold η_{idf}.

For example, "in proceedings of the" becomes a template collocation if it is not contained in another collocation and it appears sufficiently frequently (so its idf is below η_{idf}).

Since we pursue a template collocation co-occurrence mining in this work, we require additional definitions to quantify the joint occurrence of template collocations. We denote by TC_j the set of template collocations in $Val(f_j)$. For two template collocations $tc_1, tc_2 \in TC_j$, let $\rho(tc_1)$ be the frequency of tc_1 in $Val(f_j)$ and $\rho(tc_1, tc_2)$ be the pair frequency in $Val(f_j)$. They are defined as:

$$\rho(tc_1) = |\{v|v \in Val(f_j), tc_1 \text{ is a substring of } v\}|$$

$$\rho(tc_1, tc_2) = |\{v|v \in Val(f_j), tc_1 \text{ and } tc_2 \text{ are substrings of } v\}|$$

Definition 4.5. A template collocation tc_1 is an **asymmetric twin (a-twin)** of a template collocation tc_2 if it satisfies the following two conditions:

1) $\rho(tc_1, tc_2) > \rho(tc_1, tc), \forall tc \in TC_j \wedge tc \neq tc_2$, and
2) $\dfrac{\rho(tc_1, tc_2)}{\rho(tc_2)} > \eta_{tccr}$.

where η_{tccr} is the threshold.

For example, the template collocation "conference on" is an a-twin of template collocation "in proceedings of the" because it co-occurs most frequently with "in proceedings of the" and the ratio of ρ("conference on", "in proceedings of the") and ρ("in proceedings of the") is larger than threshold η_{tccr} in our dataset.

With the help of these definitions we are able to uncover "hidden" knowledge from the collection of values of a field $Val(f_j)$, which can then be used to perform value-component-level normalization for the field f_j. We base our inference on three main empirical observations. (1)Many common value components of a field are abbreviations, which need to be expanded to improve the readability of the normalized record. For example, in the field `venue`, "proc" is often used to represent "proceedings." (2)The subcollocation relation is a useful tool to organize the components of the values of a field in a partial order and then identify a template collocation from them. For example, "in proceedings of the" is a template collocation, but it oftentimes takes the form of subcollocations such as "proceedings of", "proceedings of the" and "in proceedings", which should be replaced with the template collocation. (3)Template collocations tend to co-occur frequently. For example, "conference on" frequently co-occurs with "in proceedings of the".

In this section, we present a method to mine relationships between collocations from the field values. The proposed method has three steps: (1) find pairs of the form "abbreviation and its definition" (Section 4.3.1), (2) find template collocations with their subcollocations (Section 4.3.2), and (3) find a-twin template collocations (Section 4.3.3).

4.3.1 Mining Abbreviation-Definition Pairs

We use a number of heuristics to determine whether given two value components s and t, s is an abbreviation of t. In this section, a value component is a word (or term). As we mentioned previously, in this paper we consider only fields with the string data type. We define the neighboring context of a word w within the set of values of a field f_j as the set of pairs $(left\_neighbor\_word, right\_neighbor\_word)$ with the property that the substring $left\_neighbor\_word\ w\ right\_neighbor\_word$ is a substring of a value of f_j in some record in R^e. If w is the beginning word of a field value, we use a special start-symbol "$\langle s \rangle$" to mark $left\_neighbor\_word$. If it is the last word in the field value, we use the special end-symbol "$\langle /s \rangle$" to mark $right\_neighbor\_word$. For example, the words "proceedings" and "proc" occur many times in the field `venue`, and they share a good fraction of their neighboring contexts, such as (in, of), ($\langle s \rangle$, of), (in, acm). "proc" is also the prefix of "proceedings", so we become increasingly confident that "proc" is a possible abbreviation of "proceedings". The algorithm for finding pairs of the form (s, t), where s is an abbreviation of t, is given in Algorithm 1.

Algorithm 1 Mining Abbreviation-Definition Pairs

Input: $Val(f_j) = \{r_i[f_j]|r_i \in R^e\}$: the collection of all values of the field f_j
Output: AWP: a set of abbreviation-word pairs

```
1: cwords = ∅; AWP = ∅;
2: pwords = tokenize(Val(f_j))
3: uwords = unique(pwords);
4: for each uword ∈ uwords do
5:    if len(uword) ≥ η_len and idf(uword, R^e) ≤ η_idf then
6:       insert uword into cwords;
7:    end if
8: end for
9: for each cword ∈ cwords do
10:   pa_words = getWordsBySameContext(
                    cword, uwords, η_pos);
11:   if pa_words ≠ ∅ then
12:      abbreviations = getAbbreviations(
                          cword, pa_words);
13:   end if
14:   if abbreviations ≠ ∅ then
15:      for each abbreviation ∈ abbreviations do
16:         insert (abbreviation, cword) into AWP;
17:      end for
18:   end if
19: end for
20: return AWP
```

Algorithm 1 starts with initializing two sets: $cwords$ and AWP,where $cwords$ stores the words that are likely to have abbreviations and AWP stores the final abbreviation-word pairs. In line 2, the function $tokenize$ segments all field values in $Val(f_j)$ into individual words and stores them into $pwords$. In line 3, the function $unique$ looks for unique words and stores them into $uwords$. In lines 4-8, the words in $uwords$ whose lengths are larger than a threshold η_{len} and their idfs are less than a threshold η_{idf} become candidate words with abbreviations. They are stored into $cwords$. η_{len} and η_{idf} are empirically set. For each $cword$ in $cwords$, lines 9 to 19 find its possible abbreviations. The function $getWordsBySameContext$ looks for the possible abbreviated words for each $uword$ in $uwords$. It accom-

plishes this task by measuring the size of the intersection of the neighboring contexts of $uword$ and $cword$. Then it sorts the words in descending order of the size of the intersection with the neighboring context of $uword$, retains only the top η_{pos} of them and returns them in the set pa_words. In line 12, the function $getAbbreviations$ finds the words in pa_words that are prefixes of $cword$. It returns them in $abbreviations$. For each $abbreviation$ in $abbreviations$, the pair ($abbreviation$, $cword$) is inserted into AWP. Finally, the algorithm returns AWP.

4.3.2 Mining Template Collocations and Subcollocations

Let an n-collocation tc be a template collocation and a k-collocation kc be its subcollocation ($k < n$). We observe that a number of rules govern the expansion process of kc to tc.

Rule 1. If kc is a subcollocation of a (k+1)-collocation $k1c$, and the extra word in $k1c$ is a preposition (e.g., "in" and "on") or an article (e.g., "the", "a", and "an"), we can expand kc to $k1c$.

Consider kc = "proceedings of" and $k1c$ = "proceedings of the", $k = 2$. kc is a subcollocation of $k1c$ and "the" is a preposition. Thus, "proceedings of" can be expanded to "proceedings of the." In another example, "conference on" is the subcollocation of "international conference on" and the distinct word "international" is neither a preposition nor an article, so we cannot expand "conference on" to "international conference on". Not every venue has the word "international," which suggests that this expansion is infeasible in practice.

Rule 2 (Transitivity)**.** If a k-collocation kc can be expanded to a (k+1)-collocation $k1c$ and $k1c$ can be expanded to a (k+2)-collocation $k2c$, then kc can be expanded to $k2c$.

For example, "proceedings of" can be expanded to "in proceedings of the" via "proceedings of the". The transitive property is an immediate consequence of Rule 1. Thus, we can use it to expand kc to tc.

Rule 3 (Start)**.** All one word collocations (i.e., $k = 1$) are nouns.

We use POS tagger in *NLTK* [22] to get the part of speeches of the words in the experimental studies.

Using the above analysis, we aim to find all template collocations and their subcollocations. The template collocations become the candidates with which we can expand (replace) the subcollocations. They will be used to generate the normalized component values for a field. The algorithm of finding template collocations and its subcollocations is given in Algorithm 2.

The input of the algorithm $CVal(f_j)$ is the updated version of $Val(f_j)$, the collection of all values of the field f_j, where the abbreviations are extended by Algorithm 1. The output is a set of pairs $TCSP$. A pair (tc, S_{tc}) in $TCSP$ denotes a template collocation tc and its set of subcollocations S_{tc}. We will use the output to replace the occurrence of an element in S_{tc} in some value of the field f_j with tc when we build the normalized record. We now describe the main steps of our mining algorithm(Algorithm 2). We set $TCSP$ to empty set and set m to the largest word (term) count encountered in any of the values in $CVal(f_j)$. m is the upper bound for the length of a template collocation; any tc in the output set $TCSP$ has at most m words. (A collocation is a substring of some value of the field f_j in some record $r \in R^e$, hence a collocation cannot exceed the largest value length –measured in the number of words– for the field f_j.) The algorithm builds the set of one-word collocations, according to Rule 3. If this set is empty, the algorithm stops because there are no nouns and we cannot construct any meaningful collocations. Otherwise, the set of one-word collocations are used to seed $TCSP$. We also extract the set of words (prepositions and articles) which help construct collocations of larger lengths (according to Rule 1). The main body of the algorithm is in the for loop (Lines 10- 28). In iteration $n, 2 \leq n \leq m$, the algorithm performs the following main computational steps:

Algorithm 2 Mining TemplateCollocation-SubCollocation Pairs (MTS)

Input: $CVal(f_j)$ – the updated version of $Val(f_j)$ with abbreviations extended by Algorithm 1.
Input: η_{idf}.
Output: $TCSP$: a set of pairs$\{(tc, S_{tc})\}$, where tc is a template collocation and S_{tc} its subcollocations.

1: $TCSP = \emptyset$; m=*getMaxWordCount*($CVal(f_j)$);
2: 1-*collocs* = *getOneWordCollocations*($CVal(f_j)$); //Rule 3
3: **if** 1-$collocs == \emptyset$ **then**
4: **return** $\emptyset$
5: **end if**
6: **for each** 1-$colloc \in$ 1-$collocs$ **do**
7: add (1-$colloc$, $\emptyset$) to $TCSP$;
8: **end for**
9: ews = *getCandidateExpandWords*($CVal(f_j)$)); //Rule 1
10: **for** $n = 2$ to m **do**
11: n-$collocs$ = *getNCollocations*($CVal(f_j), n, \eta_{idf}$);
12: **if** n-$collocs == \emptyset$ **then**
13: **break**;
14: **end if**
15: $Y = \emptyset$; //pairs to be ignored
16: **for each** n-$colloc \in n$-$collocs$ **do**
17: $cspairs$ = *getExpandedSubcollocationPairs*(n-$colloc$, ews, $TCSP$);
18: **if** $cspairs \neq \emptyset$ **then**
19: **for each** $cspair \in cspairs$ **do**
20: {$cspair$ is of the form (c, S_c), c is a collocation and S_c its set of subcollocations; c is a subcollocation of n-$colloc$}
21: $X = \{c\} \cup S_c$;
22: insert (n-$colloc$, X) into $TCSP$;
23: add $cspair$ to Y; //not a template collocation
24: **end for**
25: **end if**
26: **end for**
27: $TCSP = TCSP - Y$;
28: **end for**
29: remove the pairs of the form (c, $\emptyset$) from $TCSP$;
30: **return** $TCSP$

- it constructs all collocations of n words, i.e., n-collocations, according to Definitions 4.2 and 4.3, using Rule 1 (Line 11).
- for each $n-collocation$ $n-colloc$, it identifies all the entries $(c, S_c) \in TCSP$ with the property that c is a subcollocation of $n-colloc$. They are denoted $cspairs$ in the algorithm. The set union X of their S_c's (subcollocations) along with all c's is attached to $n-colloc$ and inserted in $TCSP$, according to the transitivity property in Rule 2 (Lines 20 - 23). The intuition is that $n-colloc$ is a candidate template collocation that can replace all the collocations in X.
- it removes the entries (c, S_c) from $TCSP$ from the previous step because they cannot be template collocations, based on Definition 4.4 (Lines 23 and 29).
- it may exit the for loop earlier if it cannot construct collocations of length n, $n < m$ (Lines 12 - 14).

Before termination, the algorithm removes all the pairs $(c, \emptyset)$ form $TCSP$. These are the pairs introduced in the intialization step, but never expanded by the main body of the algorithm.

4.3.3 Frequent Template Collocation Mining

In Section 4.3.2, we discussed how to obtain the template collocations and their corresponding subcollocations. We notice that some of the template collocations co-occur frequently. For example, among the values of the field `venue`, the template collocation "conference on" co-occurs most frequently with "in proceedings of the." We also observe that template collocation co-occurrence is not always bidirectional. For example, the template collocation "symposium on" co-occurs most often with "in proceedings of the", but "in proceedings of the" co-occurs most frequently with "conference on." This justifies our choice of an asymmetric co-occurrence measure in Definition 4.5. We give here an algorithm (Algorithm 3) for finding most frequently co-occurring template collocations (the a-twins).

The input of the algorithm is the collection of all values of the field f_j and the output is a set of pairs T_{atwin} in which each pair is in the form (tc_1, tc_2), where tc_1 is an a-twin of tc_2 (Definition 4.5). We start by updating $Val(f_j)$ with the findings about abbreviations: the function *updateValWithAWP* calls Algorithm 1. We then compute all template collocations TC_j for field f_j (Lines 3 - 4); the function MTS invokes Algorithm 2 to gather the template collocation. Next, we compute the frequencies of occurrence of each template based on TC_j and $CVal(f_j)$ (Line 5). The set $\overline{TC_j}$ contains pairs of the form (tc_1, ρ_1), where ρ_1 is the frequency of tc_1 in $CVal(f_j)$. The main body of the algorithm is in the for loop (Lines 6 - 13), which computes the a-twin of each template collocation in TC_j. For a template collocation $tc_1 \in TC_j$, it first finds its most frequent co-occurring template tc_2 (Definition 4.5, 1)). Then, it checks the second condition in Definition 4.5 (Lines 8 - 12). If tc_2 meets both conditions, the pair is appended to T_{atwin}.

4.3.4 Complexity Analysis of Algorithms

In this section, we provide complexity analysis of the above three algorithms. Let n denote the number of entities of a dataset, ne denote the average number of matching records per entity, nf denote the average number of fields per record, and mw denote the largest number of words in a field.

Algorithm 3 Mining Most Frequently Co-occurring Template Collocation

Input: $CVal(f_j) = \{r_i[f_j] | r_i \in R^e\}$: the collection of all values of field f_j
Input: η_{tccr}
Output: T_{atwin}: the set of most frequently co-occurring pairs of template collocations

1: $T_{atwin} = \emptyset$;
2: $CVal(f_j)$ = *updateValWithAWP*($Val(f_j)$);
3: Z=*MTS*($Val(f_j)$); //Z has pairs of the form (tc, S_{tc})
4: TC_j=*getTemplateCollocations*(Z); //TC_j is the set of tc's
5: $\overline{TC_j}$=*getTCPCounts*($TC_j, CVal(f_j)$);
6: **for each** $tc_1 \in TC_j$ **do**
7: (tc_2, ρ)=*getMostFrequentTwinTC*($tc_1, TC_j, CVal(f_j)$);
8: ρ_2 = *getCount*(tc_2, $\overline{TC_j}$);
9: $ratio$=$\frac{\rho}{\rho_2}$;
10: **if** $ratio > \eta_{tccr}$ **then**
11: insert (tc_1, tc_2) into T_{atwin}; //Definition 4.5 2)
12: **end if**
13: **end for**
14: **return** T_{atwin}

Algorithms 1-3 are for processing one field (f_j) of all records. In reality a record has multiple fields, so the computational complexities of Algorithms 1-3 all need to be multiplied by nf.

In Algorithm 1, functions $tokenize$ in line 2 and $unique$ in line 3 both need to go through all values of the field f_j, so their time complexity is $O(n \times ne \times mw)$. In lines 4 to 8, for each $uword$ in $uwords$, we judge if it is a candidate word with abbreviations. In the worst case, line 6 is within time $O(n \times ne \times mw)$. In lines 9 to 19, for each $cword$ in $cwords$, we find its possible abbreviation. As function $getWordsBySameContext$ needs to go through every $uword$ in $uwords$ and function $getAbbreviation$ needs to scan every words in pa_words, the worst case of line 10 and line 12 are both within time $O(n^2 \times ne^2 \times mw^2)$. The running time of line 16 depends on the size of $abbreviations$, so the worst case of line 16 is also within time $O(n^2 \times ne^2 \times mw^2)$. Thus the time complexity of Algorithm 1 is at most $O(n^2 \times ne^2 \times mw^2)$.

The time complexity of Algorithm 2 depends on that of line 19 which is the innermost loop. The running time of line 19 is $mw \times |n\text{-}collocs| \times |cspairs|$ where $|n\text{-}collocs|$ and $|cspairs|$ denote the size of $n\text{-}collocs$ and $cspairs$, respectively. In the worst case, $|n\text{-}collocs|$ and $|cspairs|$ are both close to $n \times ne \times mw$. Thus the time complexity of Algorithm 2 is at most $O(n^2 \times ne^2 \times mw^3)$.

In Algorithm 3, as it invokes Algorithm 2 in line 3, so its time complexity is at least as large as that of Algorithm 2. Functions $getTemplateCollocations$ and $getTCPCounts$ both need to go through $CVal(f_j)$, the time of each line is $O(n \times ne \times mw)$. In lines 6 to 13, for each tc_1 in TC_j, we find a-twin of each template collocation in TC_j. As

functions $getMostFrequentTwinTC$ and $getCount$ both need to scan $CVal(f_j)$, the worst case of lines 7 and 8 are both within $O(n^2 \times ne^2 \times mw^2)$ which is smaller than the running time of line 3. Thus the time complexity of Algorithm 3 is at most $O(n^2 \times ne^2 \times mw^3)$.

4.4 Ranked List Merging

In Section 4.2, we introduced a set of single-strategy rankers each of which ranks the units (records or field values) with a different strategy. In general, a single-strategy approach does not produce satisfactory results and may even cause bias. We utilize a multi-strategy approach to combine the outcomes of several single-strategy rankers to overcome the limitations of the individual rankers. A multi-strategy approach requires an effective *rank merging algorithm* [3].

Suppose that we have M single-strategy rankers. Denote by L_i the ranked list of units produced by the i^{th} ranker on a set of units U. The problem is that of creating a single ranked list L of U using the ranking information supplied by the individual rankers. This task is called *result merging* [3], [23], [24] and merging based on local ranks is the class of merging algorithms most frequently employed for this task. We employ two merge algorithms from this class based on the Borda-fuse method [25]. We describe them below.

4.4.1 Borda-based Approach

Let $|U|$ be the number of units U. In the classic Borda-fuse approach, the first ranked unit in each L_i gets the score $|U|$, the second ranked unit gets the score ($|U| - 1$ the second ranked unit gets score ($|U| - 1$), and so on. The units in the merged list are ranked in descending order of the sum of their scores across all L_i's. The unit with the largest combined score becomes the normalized unit (record or field value). This approach utilizes the position information in every ranked list, but one of its weaknesses is that it treats uniformly the individual rankers. In general, some rankers are better than others in suggesting normalized units.

4.4.2 Weighted-Borda-based Approach

This approach attempts to differentiate the impact of each ranker by assigning a weight to each ranker. The weight represents our belief in the quality of the suggested normalized unit by the ranker. We propose two methods to compute the weights of the individual rankers. The first method applies k-fold cross-validation on the training dataset for each ranker, and takes the average precision of a ranker as its weight. The second method uses a genetic algorithm to train a weight vector with the number of rankers over the training dataset to obtain the optimal weights. We tested both methods and the second method yielded better performance. In the rest of this paper, we use the weights obtained with the second method. After we compute the weight of each ranker, we compute the aggregated weighted score of each unit over all lists L_i. The unit with the largest aggregated weighted score is selected as the normalized one.

TABLE 2
Instances of previously used gold standard venue values [26] and of gold standard venue values according to our manual annotation

id	Old gold standard	New gold standard
1	in international conference on database theory	in **proceedings of the 3rd** international conference on database theory
2	in proceedings sixth international conference on network protocols	in proceedings **of the 6th** international conference on network protocols
3	in proceedings of 1st int conf on audio and video based biometric person authentication	in proceedings of **the** 1st **international conference** on audio and video based biometric person authentication

5 Experiments

5.1 Dataset

We use the dataset PVCD [26]. The dataset contains data about publication venue canonicalization [27]. PVCD has 3,683 publication venue values for 100 distinct real-world publication records. It is only concerned with the field `venue`, which is arguably the most difficult field to normalize, because of the presence of acronyms, abbreviations, and misspellings. We use this dataset to compare our approaches with those in [26]. The work in [26] is an instance of typical normalization, because it selects one of the duplicate records or one of the field values as the normalized record or field value, respectively. It does not attempt to create new field values or new records as normalized records. Our analysis of the dataset reveals that many normalized field values are labelled unreasonably. We point out some of the problems in Table 2. The column "old gold standard" shows the normalized `venue` values as used in the experimental study of Culotta et al. [26] and the column "new gold standard" shows them after we curated the dataset.

As Table 2 illustrates, many of the "old" gold standard field values are incomplete, missing key value components, such as "proceedings of the [ordinal number]". The second row of the table shows that many other old gold standard values miss the value component "of the". The third row in the table points out instances that miss the value component "the" and that acronyms are not expanded, e.g., "int" and "conf" are not expanded to "international" and "conference", respectively. In this paper, we will perform value-component-level normalization and compare against the new, corrected gold standard. For ease of reference, we refer to the dataset used in [26] as O-PVCD and to the one that we manually adjusted as N-PVCD in this section. The data is available at https:/github.com/tomdyq/RecordNormalization/tree/master/data.

We perform 5-fold cross validation on the data; each split contains 80 training samples and 20 testing examples. We implement eight different normalization techniques corresponding to the methods described in Section 4.

5.2 Performance Metrics

We measure accuracy by taking the proportion of correct normalized units (records or field values) out of all pre-

TABLE 3
The accuracy of our normalization methods on the dataset N-PVCD

Category	Approach	FL Typical	VCL Complete
single-strategy	Frequency Ranker(FR)	0.18	0.68
	Length Ranker(LR)	0.12	0.34
	Centroid Bigram Ranker(C_BR)	0.25	0.75
	Centroid Winkler Ranker(C_WR)	0.24	0.76
	Centroid Edit-distance Ranker(C_EDR)	0.28	0.81
	Feature-based Ranker(FBR)	0.31	0.72
multi-strategy	Borda	0.28	0.79
	Weighted Borda(WBorda)	**0.33**	**0.83**

dicted normalized units. We have three accuracy measures: record-level, field-level and value-component-level. As the dataset only has one field, the accuracies of the first and second levels are the same. Hence, we only report the field-level (FL) and value-component-level (VCL) accuracies.

5.3 Experimental Results

We perform five experiments to evaluate the effectiveness of our approach.

5.3.1 Main Experimental Results

Table 3 summarizes the outcome of our eight approaches for the N-PVCD dataset. The first six rows in the table belong to the category of single-strategy approaches and the last two rows belong to the multi-strategy approaches. We will use the acronyms in parenthesis to refer to these approaches for the rest of this section.

The main conclusion of this experimental study is that WBorda consistently outperforms the other approaches on both FL typical normalization and VCL complete normalization. For single-strategy, FBR (Feature-based Ranker) has the best accuracy on these two forms of normalization. WBorda outperforms FBR by 6.5% on FL typical normalization and by 15.3% on VCL complete normalization. We find that the accuracy of Borda is lower than that of FBR on FL typical normalization, but higher than that of FBR on VCL complete normalization. Our explanation is that Borda treats uniformly the rankers and some rankers may have poor performance, which affects the final result. When rankers are assigned weights according to their contributions to the normalized record, WBorda significantly improves the normalization accuracy.

We notice that FL typical normalization appears to give very low accuracy. The reason is that many publication entities in N-PVCD have no record in their group of matching records that contains the normalized field value. We have computed the ratio of the publication entities without normalized field values in our annotation in each fold of the cross validation. The results are shown in Table 4. As shown in the table, in each fold of the cross validation, more than half of the publication entities lack a normalized field value. The average ratio of entities without normalized field values reaches 0.63. So the maximum possible average accuracy that can be achieved is 0.37. Thus the accuracy of 0.33 achieved by WBorda is quite close to the theoretical maximum average accuracy (close to 90%).

TABLE 4
The ratio without normalized field value on N-PVCD

round of 5-fold cross validation	1	2	3	4	5
ratio of the entities without normalized field value	0.6	0.75	0.65	0.55	0.6
average ratio of the entities without normalized field value	0.63				

TABLE 5
Comparison with the baseline approach on typical normalization

Dataset	Baseline Accuracy	Our Accuracy
O-PVCD	0.6	0.65
N-PVCD	0.28	0.33

5.3.2 Comparison with the Baseline

We compare our results with the approach in [26], which serves as the baseline, on the datasets O-PVCD and N-PVCD. The work in [26] performed only typical normalization, while we perform both typical and complete normalizations. In this experimental study, we use our best performing method, which is WBorda. The source code of the approach in [26] is not publicly available. We implemented the best method reported by Callota et al. [26] to the best of our understanding. The outcome of this experimental study is given in Table 5.

Our approach outperforms the baseline by a significant margin: by 8.3% on O-PVCD and by 17.9% on N-PVCD. The reason for the seemingly low accuracy on N-PVCD of the two techniques was given in Section 5.3.1.

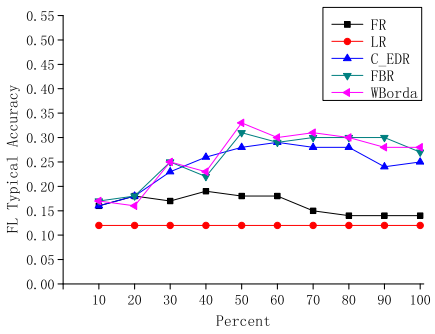
We additionally compare the baseline and our method on the new gold standard N-PVCD, for the complete normalization. Since the baseline cannot carry out a complete normalization, we use our implementation of the baseline approach to perform the FL typical normalization and use the same mined knowledge to complete the field value. The result is shown in Table 6. Our approach outperforms the baseline again by a significant margin, 12.2%.

TABLE 6
Comparison with the baseline approach on complete normalization

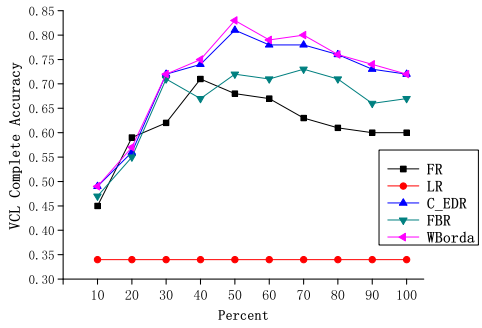
Dataset	Baseline Accuracy	Our Accuracy
N-PVCD	0.74	0.83

5.3.3 Impact of the Percent of Units in Ranked List of Each Ranker in the Multi-strategy Approach

In the multi-strategy approach, each strategy ranker respectively generates a ranked list. As there are still some units (records/field values) in each ranked list that have very small probabilities of becoming a normalized unit, we perform pruning operation before rank merging. In this experiment, we evaluate the impact of the percent of units in ranked list of each ranker.



(a)FL Typical Accuracy Comparison



(b)VCL Complete Accuracy Comparison

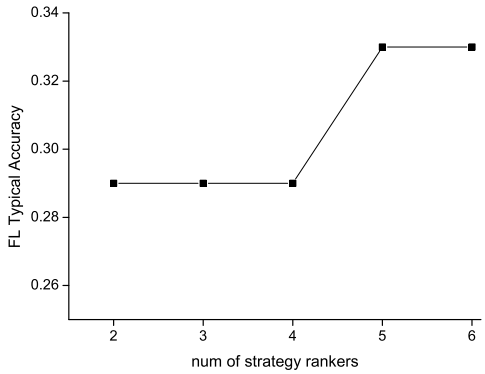
Fig. 3. Performance comparison by different approaches on different percent of ranked results on N-PVCD

We use the percent of ranked list of candidate units to judge which units must be kept to compute the normalized unit. We use p percent to keep the top $p\%$ and prune the remaining $(100-p)\%$ of the ranked matched units. The percent of the ranked result is varied from 10% to 100% in increments of 10% in each step. The result is shown in Fig.3. We observe that WBorda, FBR and C_EDR all reach the highest values respectively in FL typical normalization and VCL normalization at 50% of the ranked results. FR reaches the highest accuracy at about 40%. We also observe that the accuracy of LR does not change, because in this case in every percent of the ranked results, the longest field value always lies in the first position. In all our experiments, our approach is based on 50% of ranked result.

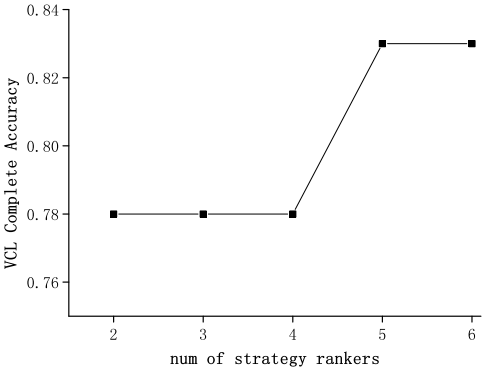
5.3.4 *Impact of Individual Rankers*

Our two rank merging methods use all the rankers in all the experiments reported above. We know that the rankers have varied accuracies. In this experiment we study the impact of the individual rankers on the overall accuracy. We use only the WBorda method because it is our best performing method. We order the rankers by their performance: FBR, C_EDR, C_BR, C_WR, FR, and LR. We analyze WBorda with the first k (k=2,3,. . . ,6) of them and report the accuracy for each k. For example, WBorda uses FBR, C_EDR and C_BR for k = 3. We use the dataset N-PVCD. Fig.4 shows the outcome of this experimental study.

We observe that WBorda with FBR and C_EDR achieves an accuracy of 0.29 on field typical normalization. C_BR and C_WR only slightly increase their performance. However,



(a)FL Typical Accuracy Change



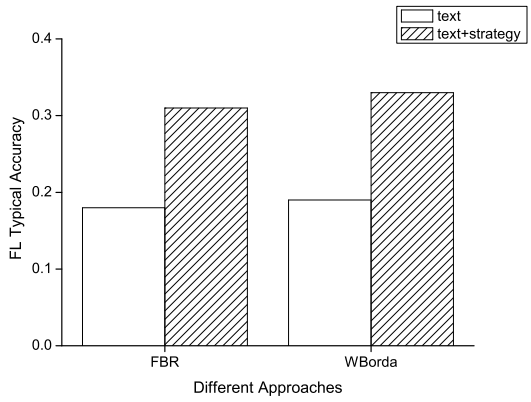
(b)VCL Complete Accuracy Change

Fig. 4. Performance Change of partially merging strategy rankers on N-PVCD

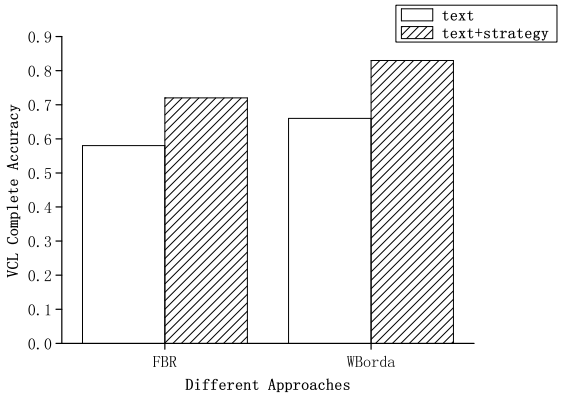
WBorda's performance increases by 0.05 if FR and LR are used. A similar pattern is observed for value component normalization. WBorda with the first three rankers achieves an accuracy of 0.78. Its accuracy increases by 0.05 with the addition of the rest of the rankers. Hence, the top-2 performing individual rankers, if combined, give the highest accuracy increase over the individual rankers. Therefore, at least for the domain of scientific publications, the rankers C_BR, C_WR and LR bring limited accuracy increase and may be dropped. They should not be discarded in general without a thorough empirical study on the domain at hand.

5.3.5 *Impact of Features on FBR and WBorda*

Feature-based Ranker (FBR) employs two types of features, strategy features and text features (described in Section 4.2.4), in this section we report their effect on FBR. At the same time, WBorda has the best performance in our experiment, which fuses FBR, so we also test the effect of these two types of features on WBorda. Fig.5 shows the performance of the two approaches with and without the strategy features. We observe that using only text features in FL typical normalization, FBR and WBorda reach accuracies of .18 and .19, respectively, while in VCL normalization, they reach accuracies of .58 and .66, respectively. Adding strategy features improves the accuracies of FBR and WBorda by .13 (or 72%) and .14 (or 74%), respectively, in FL typical



(a)FL Typical Normalization Accuracy Comparison



(b)VCL Normalization Accuracy Comparison

Fig. 5. Performance Comparison of FBR and WBorda with and without strategy features on N-PVCD

normalization and by .14 (or 24%) and .17 (or 26%) in VCL normalization, respectively.

6 Related Work

In this section, we review the literature on record normalization. We give a few pointers on the related problems of schema integration and ontology merging.

The problem of normalization of database records was first described by Culotta et al. [26]. They provided the first attempt to formalize the record normalization problem and proposed three solutions. The first solution uses string edit distance to determine the most central record. The second solution optimizes the edit distance parameters, and the third one describes a feature-based solution to improve performance by means of a knowledge base. Their approach is an instance of typical field value normalization. They did not consider value-component-level normalization. In addition, their gold standard dataset has many instances of unreasonable normalized records.

Swoosh [28] describes a record Merge operator, however, the purpose of the operator is not for producing normalized records, but rather for improving the ability to establish difficult record matchings.

Wick et al. [29] propose a discriminatively-trained model to implement schema matching, reference, and normalization jointly. But the complexity of the model is greatly increased. This paper also contains no discussion on complete normalization at the value-component level.

Besides the above works that explicitly address record normalization, a few others include (or refer to) the general idea of record normalization in some form. Tejada et al. [11] devise a system to automatically extract and consolidate information from multiple sources into a unified database. Although object deduplication is the primary goal of their research, record normalization arises when the system presents results to the user. They propose ranking the strings for each attribute based on the user's confidence in the data source from which the string was extracted. Wang et al. [30] propose a hybrid framework for product normalization in online shopping by schema integration and data cleaning. Although their work mainly focuses on record matching, they consider the problem of filling missing data and repairing incorrect data, which is relevant to record normalization. Chaturvedi et al. [31] propose an automatic pattern discovery method for rule-based data standardization systems. Their goal is to help domain experts find the important and prevalent patterns for rule writing. Although they do not directly explore the problem of record normalization, their pattern discovery approach could be used for complete normalization.

Label normalization in schema integration is related to record normalization. Dragut et al. [32] propose a naming framework to assign meaningful labels to the elements of an integrated query interface. Their approach can capture the consistency among the labels assigned to various attributes within a global interface.

Ontology merging is another area related to record normalization [33]. A domain expert is usually deeply involved during the merging process, whereas our approach strives to reduce human involvement as much as possible.

7 Conclusion and Future Work

In this paper, we studied the problem of record normalization over a set of matching records that refer to the same real-world entity. We presented three levels of normalization granularities (record-level, field-level and value-component level) and two forms of normalization (typical normalization and complete normalization). For each form of normalization, we proposed a computational framework that includes both single-strategy and multi-strategy approaches. We proposed four single-strategy approaches: frequency, length, centroid, and feature-based to select the normalized record or the normalized field value. For multi-strategy approach, we used result merging models inspired from metasearching to combine the results from a number of single strategies. We analyzed the record and field level normalization in the typical normalization. In the complete normalization, we focused on field values and proposed algorithms for acronym expansion and value component mining to produce much improved normalized field values. We implemented a prototype and tested it on a real-world dataset. The experimental results demonstrate the feasibility

and effectiveness of our approach. Our method outperforms the state-of-the-art by a significant margin.

In the future, we plan to extend our research as follows. First, conduct additional experiments using more diverse and larger datasets. The lack of appropriate datasets currently has made this difficult. Second, investigate how to add an effective human-in-the-loop component into the current solution as automated solutions alone will not be able to achieve perfect accuracy. Third, develop solutions that handle numeric or more complex values.

ACKNOWLEDGMENTS

This work was supported in part by the following grants: National Natural Science Foundation of China (No. 61100167), Natural Science Foundation of Jiangsu Province, China (No. BK2011204), and Qing Lan Project; and by the U.S. National Science Foundation BIGDATA 1546480 and 1546441, and the National Institute of Health R01, LM010817-01.

REFERENCES

[1] K. C.-C. Chang and J. Cho, "Accessing the web: From search to integration," in *SIGMOD*, 2006, pp. 804–805.
[2] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "Webtables: Exploring the power of tables on the web," *PVLDB*, vol. 1, no. 1, pp. 538–549, 2008.
[3] W. Meng and C. Yu, *Advanced Metasearch Engine Technology*. Morgan & Claypool Publishers, 2010.
[4] A. Gruenheid, X. L. Dong, and D. Srivastava, "Incremental record linkage," *PVLDB*, vol. 7, no. 9, pp. 697–708, May 2014.
[5] E. K. Rezig, E. C. Dragut, M. Ouzzani, and A. K. Elmagarmid, "Query-time record linkage and fusion over web databases," in *ICDE*, 2015, pp. 42–53.
[6] W. Su, J. Wang, and F. Lochovsky, "Record matching over query results from multiple web databases," *TKDE*, vol. 22, no. 4, 2010.
[7] H. Köpcke and E. Rahm, "Frameworks for entity matching: A comparison," *DKE*, vol. 69, no. 2, pp. 197–210, 2010.
[8] X. Yin, J. Han, and S. Y. Philip, "Truth discovery with multiple conflicting information providers on the web," *ICDE*, 2008.
[9] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *TKDE*, vol. 19, no. 1, pp. 1–16, 2007.
[10] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," *TKDE*, vol. 24, no. 9, 2012.
[11] S. Tejada, C. A. Knoblock, and S. Minton, "Learning object identification rules for information integration," *Inf. Sys.*, vol. 26, no. 8, pp. 607–633, 2001.
[12] L. Shu, A. Chen, M. Xiong, and W. Meng, "Efficient spectral neighborhood blocking for entity resolution," in *ICDE*, 2011.
[13] Y. Jiang, C. Lin, W. Meng, C. Yu, A. M. Cohen, and N. R. Smalheiser, "Rule-based deduplication of article records from bibliographic databases," *Database*, vol. 2014, 2014.
[14] X. Li, X. L. Dong, K. Lyons, W. Meng, and D. Srivastava, "Truth finding on the deep web: Is the problem solved?" in *PVLDB*, vol. 6, no. 2, 2012, pp. 97–108.
[15] J. Pasternack and D. Roth, "Making better informed trust decisions with generalized fact-finding," in *IJCAI*, 2011, pp. 2324–2329.
[16] X. L. Dong and F. Naumann, "Data fusion: resolving data conflicts for integration," *PVLDB*, vol. 2, no. 2, pp. 1654–1655, 2009.
[17] E. K. Rezig, E. C. Dragut, M. Ouzzani, A. K. Elmagarmid, and W. G. Aref, "ORLF: A flexible framework for online record linkage and fusion," in *ICDE*, 2016, pp. 1378–1381.
[18] X. Wang, X. L. Dong, and A. Meliou, "Data x-ray: A diagnostic tool for data errors," in *SIGMOD*, 2015, pp. 1231–1245.
[19] G. R. D. Patrick AV Hall, "Approximate string matching," *ACM Computing Surveys*, vol. 12, no. 4, pp. 381–402, 1980.
[20] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string metrics for matching names and records," in *KDD workshop on data cleaning and object consolidation*, 2003, pp. 73–78.
[21] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, no. 3, pp. 503–528, 1989.
[22] "Natural language toolkit," http://www.nltk.org.
[23] E. Dragut, B. DasGupta, B. P. Beirne, A. Neyestani, B. Atassi, C. Yu, and W. Meng, "Merging query results from local search engines for georeferenced objects," *TWEB*, vol. 8, no. 4, 2014.
[24] J. Yuan, L. He, E. C. Dragut, W. Meng, and C. Yu, "Result merging for structured queries on the deep web with active relevance weight estimation," *Inf. Sys.*, vol. 64, pp. 93 – 103, 2017.
[25] J. A. Aslam and M. Montague, "Models for metasearch," in *SIGIR*, 2001, pp. 276–284.
[26] A. Culotta, M. Wick, R. Hall, M. Marzilli, and A. McCallum, "Canonicalization of database records using adaptive similarity measures," in *SIGKDD*, 2007, pp. 201–209.
[27] "canonicalization data," http://cs.iit.edu/~culotta/data/canonicalization.html, accessed: 2017-01-03.
[28] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom, "Swoosh: A generic approach to entity resolution," *VLDBJ*, vol. 18, no. 1, pp. 255–276, 2009.
[29] M. L. Wick, K. Rohanimanesh, K. Schultz, and A. McCallum, "A unified approach for schema matching, coreference and canonicalization," in *SIGKDD*, 2008, pp. 722–730.
[30] L. Wang, R. Zhang, C. Sha, X. He, and A. Zhou, "A hybrid framework for product normalization in online shopping," in *DASFAA*, vol. 7826, 2013, pp. 370–384.
[31] S. Chaturvedi and et al., "Automating pattern discovery for rule based data standardization systems," in *ICDE*, 2013.
[32] E. C. Dragut, C. Yu, and W. Meng, "Meaningful labeling of integrated query interfaces," in *VLDB*, 2006, pp. 679–690.
[33] S. Raunich and E. Rahm, "Atom: Automatic target-driven ontology merging," in *ICDE*, 2011, pp. 1276–1279.

Yongquan Dong received the BS degree and PhD degree in computer science from Shandong University. He is currently an associate professor in the School of Computer Science and Technology, Jiangsu Normal University, China. His research interests include Web information integration and Web data management.

Eduard C. Dragut received the PhD degree in computer science from the University of Illinois at Chicago in 2010. He is currently an assistant professor in the Department of Computer and Information Sciences, Temple University. His research interests lie in the broad area of web data management. He is a coauthor of the book *Deep Web Query Interface Understanding and Integration*. He co-chaired the VLDB QDB 2012, and the Ph.D. Symposiums at ICDE 2014 and SIGMOD/PODS 2016. He is an IEEE member.

Weiyi Meng received the BS degree in mathematics from Sichuan University, China, in 1982, and the MS and PhD degrees in computer science from the University of Illinois at Chicago, in 1988 and 1992, respectively. He is currently a professor in the Department of Computer Science at the State University of New York at Binghamton. His research interests include web-based information retrieval, metasearch engines, and web database integration. He is a coauthor of three books *Principles of Database Query Processing for Advanced Applications*, *Advanced Metasearch Engine Technology*,and *Deep Web Query Interface Understanding and Integration*. He has published more than 150 technical papers. He is a senior member of the IEEE.