Learning Top-k Transformation Rules*

Sunanda Patro and Wei Wang
{sunandap, weiw}@cse.unsw.edu.au

University of New South Wales

Abstract. Record linkage identifies multiple records referring to the same entity even if they are not bit-wise identical. It is thus an essential technology for data integration and data cleansing. Existing record linkage approaches are mainly relying on similarity functions based on the surface forms of the records, and hence are not able to identify complex coreference records. This seriously limits the effectiveness of existing approaches.
In this work, we propose an automatic method to extract top-k high quality *transformation rules* given a set of possibly coreferent record pairs. We propose an effective algorithm that performs careful local analyses for each record pair and generates candidate rules; the algorithm finally chooses top-k rules based on a scoring function. We have conducted extensive experiments on real datasets, and our proposed algorithm has substantial advantage over the previous algorithm in both effectiveness and efficiency.

1 Introduction

Real data are inevitably noisy, inconsistent, or contain errors. For example, there are *dozens* of correct ways to cite a publication, depending on the bibliography style one uses; however, there can be hundreds of citations to the same publication that contain errors, as caused by typographical errors, Optical Character Recognition (OCR) errors, or errors introduced when the citation was extracted by a program from Web pages.

Record linkage is the process of bringing together two or more separate records pertaining to the same entity, even if their surface forms are different. It is a cornerstone to ensure high quality of mission-critical data either in a single database or during data integration from multiple data sources. Therefore, it has been used in many applications including data cleansing, data integration from multiple sources or the Web, etc.

Most existing methods for record linkage rely on similarity functions to generate candidate pair of records that may be coreferent. This is insufficient when coreferent records has little surface similarity. For example, `23rd` and `twenty-third`, `VLDB` and `Very Large Databases`. Therefore, existing systems

* This work was partially supported by ARC Discovery Projects DP0987273 and DP0881779.

incorporate *transformation rules* to recognize these domain-specific equivalence relationships.

Traditionally, transformation rules were manually created by experts. This process is not only tedious, expensive, and often erroneous, the generated rules are seldom comprehensive enough. This is the main motivation for semi-automatic methods to automatically learn a set of high quality candidate rules [1]; domain experts can then manually validate or refine the candidate rules. Hence, it is important that a majority of the candidate rules generated by these algorithms should be correct. The rule learning algorithm should also be able to cope with large input datasets and learn top-k rules efficiently. As we demonstrate in the experiments, existing methods [1] fail to meet both requirements.

In this paper, we propose a novel method to automatically learn top-k transformation rules from a set of input record pairs known to be coreferent. We perform meticulous local alignment for each pair of records by considering a set of commonly used edit operations. We then generate a number of candidate rules based on the optimal local alignment. Statistics of the candidate rules are maintained and aggregated to select the final top-k rules. We have conducted extensive experiments with the existing state-of-the-art algorithm. We found that our rule learning algorithm outperforms the existing method in both effectiveness and efficiency.

Our contributions can be summarized as:

- We proposed a local alignment-based rule learning algorithm. Compared with the global greedy algorithm in [1], our algorithm generates fewer candidate rules, and our candidate rules are more likely to be correct rules.
- We have performed extensive experiments using several publicly available real-world datasets; our experimental results shows a 3.3x increase in the percentage of correct rules as compared with previous approach, and up to 300x speed-up in efficiency.

The rest of the paper is organized as follows: Section 2 introduces related work. We present our algorithm in Section 3. Experimental results are given in Section 4, and Section 5 concludes the paper.

2 Related

Record linkage is known under many different names, including entity resolution, and near duplicate detection. It is a well studied problem and has accumulated a large body of work. We refer readers to surveys [2, 3], and focus on related work that is most closely related to our proposal.

Most existing record linkage approaches exploit similarities between values of intrinsic attributes. Many similarity or distance functions have been proposed to model different types of errors. For example, edit distance is used to account for typographical errors and misspellings [4]. Jaro-Winkler distance is designed for comparing English names [5]. Soundex is used to account for misspellings due to similar pronunciations. Jaccard similarities or cosine similarities measure the

similarity of multi-token strings [6]. Similarity functions can also be weighted. A common heuristic is to use the well-known *tf-idf* scheme. To allow misspellings, a soft version of *tf-idf* is proposed in [7], and later extended in [8]. Alternatively, weights can be learned automatically using machine learning techniques [9–11]. Multiple similarity functions can be used on the same or different attributes of the records. The similarity values can either be simply aggregated or be treated as a relational input to a classifier [10].

Dealing with Complex Coreferent Records It is well-known that a single similarity function is not sufficient to identify complex coreferent records. Past efforts can be categorized into several categories below.

The first category is to learn a good similarity function using machine learning techniques [12, 13, 11]. For example, [14] employs a two-phase approach. In the first phase, attribute value matchers are tuned, and in the second phase, a SVM classifier is learned from a combination of the tuned matchers generated from the previous step. The experimental results show that trainable similarity measures are capable of learning the specific notion of similarity that is appropriate for a specific domain. While this approach focuses on homogenous string-based transformation, [15] uses heterogeneous set of models to relate complex domain specific relationships between two values. One technical issue for these learning-based approach is the selection of training datasets, especially the negative instances. [10] employs active learning techniques to minimize the interaction with users, and is recently improved by [16].

Another category of approaches is to model complex or domain-specific transformation rules [17, 1]. We compare with them in more detail in the next subsection. The learned rules can be used to identify more coreferent pairs [18, 19].

Note that although only few works in record linkage focus on transformation rules, they have been widely employed in many other areas, and automatic rule learning algorithms have been developed accordingly. In Natural Language Processing (NLP), [20] finds sets of synonyms by considering word co-occurrences. Later, [21] utilizes the similar idea to identify paraphrases and grammatical sentences by looking at co-occurrence of set of words.

Recently, researchers have used transformation rules to deduplicate URLs without even fetching the content [22, 23], e.g., `http://en.wikipedia.org/?title=*` and `http://en.wikipedia.org/wiki/*` always refer to the same web page. However, the rules and their discovery algorithms are heavily tailored for URLs.

Another closely related area is the substitution rules used in *query rewriting* [24, 25]. For example, when a user submits a query `apple music player` to a search engine, it may change the query to `apple ipod`. The substitution rules are mainly mined from query logs, and the key challenges are how to find similar queries and how to rank them.

Existing Transformation Rule Learner [1] is a recent work to learn top-k transformation rules given a set of coreferent record pairs. it has shown much better accuracy and scalability to larger datasets than the previous approach [17].

The idea of [1] is to identify candidate rules from the unmapped tokens of each record pair, and then compute the aggregated score of the candidate rules to find the top-k high quality rules. While our proposal follows the similar framework, there are a few major differences:

- We perform more refined local alignment for each record pair. We use a set of common edit operations (including typographical errors and abbreviations) to identify more mappings tokens, and hence produce fewer number of candidate rules from unmapped tokens. E.g., in Figure 1, all yellow and green tokens will be unmapped in [1]'s approach, and candidate rules which are essentially all possible mapping among them will be generated and counted. [1]'s strategy of pairing of all possible subsets of unmapped tokens not only decreases the quality of the final top-k rules, but also slows down its rule learning speed substantially.
- We can optionally support partitioning the records into fields (i.e., partitions), which further reduces the number of candidate rules, and speeds up the computation.
- We use a better scoring function than [1], which is less impacted by the length of the rules and counts the frequency of the rules by the clusters they appeared.

As demonstrated in our experiments (Section 4), these differences result in substantial improvements of our proposed algorithm over [1] in both the effectiveness and efficiency.

3 The Local-Alignment-based Algorithm

Similar to [1], the input to transformation rule learning algorithms is a set of coreferent record pairs. The overall idea of our new algorithm is to perform careful *local alignment* for each record pair first, generate *candidate rules* from the optimal local alignment, and aggregate the "strength" of the rules over all input pairs to winnow high-quality rules from all the candidate rules.

3.1 Preprocessing the Input Data

We perform the standard preprocessing for input record pairs: we remove all non-alphanumeric characters except spaces, and then converting characters to lower cases. We do not use the case information as it is not reliable for noisy input data. We then tokenize the strings into a sequence of tokens using white space as the separator. We also remove stopwords.

3.2 Segmentation

The goal of segmentation is to decompose a record into a set of semantic constituents known as *fields*, i.e., a substring of tokens in a tokenized record. For example, bibliography records can usually be segmented into the `authors`, `title`, and `venue` fields. Our framework does not rely on a specific type of segmentation method. One can use either supervised methods (e.g., CRF [26]) or simple

rule-based segmentation methods (e.g., based on punctuations in the raw input records).

Although this step is optional, appropriate segmentation is beneficial to generating better rules and faster execution of the algorithm. This is mainly because (i) We do not consider mappings for two tokens in different fields in two records. Hence fewer candidate rules are generated, and this speeds up the algorithm. (ii) Most of the rules generated across fields are actually erroneous. (iii) It is possible that we can use different parameter settings to learn rules for a particular field. For example, transformation rules for author names (e.g., omitting the middle name) probably do not apply on paper titles. We do not explore this option in this work and leave it for future work.

3.3 Local Alignment

We perform field alignment and then find the optimal local alignment between the values of the corresponding fields.

Computing the Optimal Local Alignment

We need to find a series of edit operations with the least cost to transform one string to another. We analyzed common transformations and decided that we support the following set of common edit operations:

- **Copy**: copy the token exactly.
- **Abbreviation**: allows one token to be a *subsequence* of another token, e.g., `department` ⇔ `dept`.
- **Initial**: allows one single-letter token to be equivalent to the first letter of another token, e.g., `peter` ⇔ `p`.
- **Edit**: allows the usual edit operations (i.e., Insertion, Deletion, and Substitution), e.g., `schütze` ⇔ `schuetze`
- **Unmapped**: tokens that are not involved in any transformation are denoted as *unmapped* tokens.

In addition to assigning cost to each of the above edit operations, we also prioritize them as *copy* > *initial* > *abbreviation* > *edit* > *unmapped*. In other words, if a high priority operation can convert one token to another, we do not consider operations of lower priorities. For example, between two tokens `department` and `dept`, since an *abbreviation* operation can change one into another, we do not consider *edit* operations. The cost of *copy* is always 0, and the cost of *unmapped* is always higher than other costs; the cost of other operations are subject to tuning. Algorithm 1 performs such local alignment and returns the minimum cost between two strings.

Example 1. We show two strings S and T (i.e., there is no segmentation), and the optimal local alignment in Figure 1.

Algorithm 1: AlignStrings(S, T)

```
1 c ← 0;
2 for each token w ∈ S do
3 |   mincost ← min{cost(w, w′) | w′ ∈ T};
  |   /* follows the priorities of edit operations                    */
4 |   if w is not unmapped then
5 |   |   c ← c + mincost;
6 for each unmapped token w ∈ S ∪ T do
7 |   c ← c + unmapped_cost(w);
8 return c
```

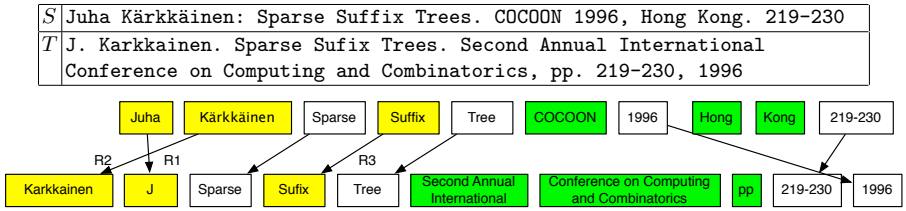


Fig. 1. Optimal Local Alignment (White cells are copied, green cells are unmapped, and yellow cells involve other types of edit operations)

Aligning Fields

If the input records have been partitioned into fields, we also need to align the fields. In the easy case where each field has a class label (as those output by the supervised segmentation methods), the alignment of fields is trivial — we just align fields with the same class label (e.g., `authors` to `authors`). If fields do not have class labels, we use Algorithm 2 to find the optimal alignment — an alignment such that the total cost is minimum. This is done by reducing the problem into a maximum weighted bipartite graph matching problem, which can be efficiently solved by the Hungarian algorithm [27] in $O(V^2 \log V + VE) = O(B_{max}^3)$, where B_{max} is the maximum number of fields in the records.

Algorithm 2: AlignBlocks(X, Y)

```
1 Construct a weighted bipartite graph G = (A∪B, E, λ), such that there is a 1-to-
  1 mapping between nodes in A and the fields in X, and there is a 1-to-1 mapping
  between nodes in B and fields in Y, E = {e_ij} connects A_i and B_j, and the
  weight of an edge is the negative of its cost, i.e., λ(e_ij) = −AlignStrings(A_i, B_j);
2 M = FindMaxMatching(G) ;                    /* use the Hungarian alg */
3 return M
```

Multi-token Abbreviation

Some of the tokens are unmapped because they involve in *multi-token abbreviation*. We generalize the *abbreviation* operation to include multiple-token to one token abbreviation. For example, in Figure 1, one such instance is `Conference on Computing and Combinatorics` to `cocoon`.

We classify multi-token abbreviations into the following categories:

- **Acronym**: A set of tokens $[u_1, u_2, \ldots, u_k]$ maps to a token v via an acronym mapping, if there exists a sequence of prefix length l_i for each u_i, such that $u_1[1, l_1] \circ u_2[1, l_2] \circ \ldots u_k[1, l_k] = v$, where $\circ$ concatenates two strings. E.g., `association computing machinery` $\Leftrightarrow$ `acm`.
- **Partial Acronym**: A set of tokens $[u_1, u_2, \ldots, u_k]$ maps to a token v via a partial acronym mapping, if there exists a prefix or suffix, v_s, of v longer than a minimum length threshold, and a subsequence $ss(u_i)$ of for each u_i, such that $ss(u_1) \circ ss(u_2) \circ \ldots ss(u_k) = v_s$. In other words, v_s is equal to a subsequence of the concatenated string $u_1 \circ \ldots \circ u_k$. E.g., `conference on knowledge discovery and data mining` $\Leftrightarrow$ `sigkdd`.

Algorithm 3: MultiToken-Abbreviation(X, Y)

Input : X and Y are the input record pairs

for each *remaining unmapped token* $v \in X$ **do**
  for each *remaining contiguous set of unmapped tokens* u *in* Y **do**
    $str \leftarrow u_1 \circ \ldots \circ u_k$;
    if *exists a prefix or suffix,* v_s*, of* v *such that* v_s *is a subsequence of* str **then**
      $partialAcronym \leftarrow$ **true**;
      if *each of the match in* u_i *starts from its first character* **then**
        $fullAcronym \leftarrow$ **true**;
    if $partialAcronym =$ **true** *or* $fullAcronym =$ **true then**
      remove matched tokens from X and Y;

The algorithm to discover both types of acronyms is depicted in Algorithm 3.

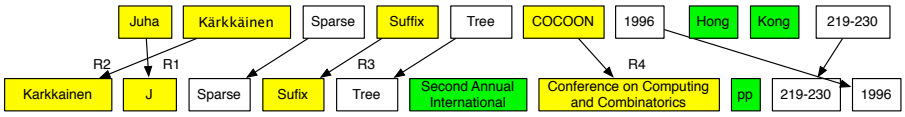


Fig. 2. Optimal Local Alignment After Recognizing Multi-token Abbreviation (R_4)

Example 2. As illustrated in Figure 2, the partial acronym mapping, denoted by R_4, is recognized.

3.4 Obtaining top-k Rules

We first generate candidate rules, compute and aggregate their scores across all input pairs of records, and finally select the top-k rules.

Definition 1 (Rule). *A rule is in the form of* `lhs` ⇔ `rhs`*, where both* `lhs` *and* `rhs` *are a sequence of tokens. The rule means that* `lhs` *is equivalent to* `rhs`*.*

Definition 2 (Atomic Rule). *An atomic rule is a rule where either the* `lhs` *or the* `rhs` *is a single token or empty (denoted as* $\perp$*).*

Note that an *omission* rule is a special rule where one side of the rule is empty. E.g., since `pp` is still unmapped in Figure 2, we have an omission rule `pp` ⇔ $\perp$.

Depending on the number of tokens on each side of the rule, we can have 1-1 rule (e.g., `peter` ⇔ `p`), 1-n rule (e.g., `vldb` ⇔ `very large databases`), n-m rule (e.g., `information systems` ⇔ `inf sys`). In practice, we observe that almost all n-m rules can be decomposed into several 1-1 or 1-n rules. E.g., the n-m rule above can be decomposed into two 1-n rules: `information` ⇔ `inf` and `systems` ⇔ `sys`. Therefore, we focus on finding atomic rules and assemble them to find more n-m rules.

Generating Candidate Rules

At this stage, we can generate rules from mapped tokens and unmapped tokens in different manners:

- For mappings (identified either by local alignment (Algorithm 1) or by multi-token abbreviation (Algorithm 3)), we can easily generate the rules. Note that we do not generate trivial copying rules (i.e., `A` ⇔ `A`). We also combine adjacent rules into n-m rules. E.g., once we identify two mappings: `information` ⇔ `inf` and `systems` ⇔ `sys`, and if `information` and `systems` are adjacent and `inf` and `sys` are adjacent too, then we also generate the rule `information systems` ⇔ `inf sys`.
- For each of the remaining unmapped tokens in one string, we postulate that it might be deleted (i.e., transformed to $\perp$), or it is mapped to every contiguous subset of the unmapped tokens in the other string. We do this for both strings. It is expected that although many of these candidate rules are invalid rules, some valid rules (usually corresponds to complex, domain-specific transformations sharing little surface similarities[1]) will appear frequently if we aggregate over large amount of input pairs.

Example 3. Based on the mappings in Figure 2, we generate the following rules from mapped tokens:

- `Juha` ⇔ `J`,
- `Käkkäinen` ⇔ `Kakkainen`,

[1] One example is `maaliskuu` ⇔ `march` found in our experiments, where `maaliskuu` in Finnish means `march` in English.

– `Juha Käkkäinen` ⇔ `Kakkainen J`,
– `Suffix` ⇔ `Sufix`,
– `COCCON` ⇔ `Conference on Computing and Combinatorics`.

Candidate rules generated from unmapped tokens `pp` are:

– `pp` ⇔ ⊥
– `pp` ⇔ `Hong`
– `pp` ⇔ `Kong`
– `pp` ⇔ `Hong Kong`

Score of a Rule

Definition 3. *We define the score of a rule R as:*

$$score(R) = \log(1 + freq(R)) \cdot \log(1 + len(R)) \cdot wt(R), \tag{1}$$

where $freq(R)$ is the number of occurrences of the rule R, $len(R)$ is the total number of tokens in R, and $wt(R)$ is the weight assigned based on the type of the rule.

The above heuristic definition takes into consideration of the popularity of a rule its length, and its type. The length component is important because whenever `ir` ⇔ `information retrieval` holds, `ir` ⇔ `information` also holds; thus *score*(`ir` ⇔ `information retrieval`) ≤ *score*(`ir` ⇔ `information`), and the partial correct rule will be incorrectly ranked higher than the complete rule. The rule type component is used to capture the intuition that, e.g., a rule generated by mapped tokens is more plausible than that generated by (randomly) pairing unmapped tokens.

Select the top-k Rules

The complete algorithm is given in Algorithm 4 which learns top-k rules with the maximum scores from a set of input pairs of records.

In Algorithm 4, Lines 1–11 generate all the candidate rules from the mappings obtained for each record pair by considering the best local alignment and possible multi-token abbreviation. Lines 12–13 calculate the scores for each candidate rule. Then we iteratively select the *TopRule* which has the maximum score (Lines 16–20), and withdraw support from other conflicting candidate rules by the procedure UpdateRules. We repeat this process until k high-quality rules are found.

The procedure UpdateRules is illustrated in Algorithm 5. We say a rule R_1 conflicts with another rule R_2 if and only if one side of R_1 is identical to one side of R_2. Two kinds of typical conflicting rules are:

– `A` ⇔ `B` and `A` ⇔ `C`
– `A` ⇔ `BCD` and `A` ⇔ `B`

Algorithm 4: Top$k$Rule($\mathcal{D}$, k)

Input : $\mathcal{D} = \{(X_1, Y_1), \ldots, (X_N, Y_N)\}$ are N input record pairs.

$\mathcal{R} \leftarrow \emptyset$;
for each *input record pair* $p_i = (X, Y)$ **do**
  $b_X \leftarrow$ segment X into partitions;
  $b_Y \leftarrow$ segment Y into partitions;
  $M \leftarrow$ AlignBlocks(b_X, b_Y);
  for each *candidate rule* R *generated from the mapping* M **do**
    if $R \notin \mathcal{R}$ **then**
      $R.support = \{p_i\}$;
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{R\}$;
    else
      Update the $R \in \mathcal{R}$ so that $R.support$ now includes p_i;
for each *candidate rule* $R \in \mathcal{R}$ **do**
  $R.score \leftarrow$ calcScore($R.support$) ; `/* based on Equation (1) */`
output $\leftarrow \emptyset$;
$i \leftarrow 1$;
while $i \leq k$ **do**
  $TopRule \leftarrow \arg\max_{R \in \mathcal{R}}\{R.score\}$;
  UpdateRules($\mathcal{R}$, *TopRule*);
  output $\leftarrow$ *output* $\cup$ *TopRule*;
  $i \leftarrow i + 1$;
return *output*

Algorithm 5: UpdateRules($\mathcal{R}$, *TopRule*)

$\{p_1, p_2, \ldots, p_l\} \leftarrow$ the support of rule *TopRule*;
for each p_i **do**
  $CR \leftarrow$ all the rules that conflict with *TopRule* from the record pair p_i;
  for each *rule* $R \in CR$ **do**
    $R.support \leftarrow R.support \setminus \{p_i\}$;

Analysis of the Algorithm Let the number of input record pairs be N, the average number of candidate rules generated by each record pair be f. Then, $|\mathcal{R}| = f \cdot N$. The time complexity of Algorithm 4 is $O(k \cdot (|\mathcal{R}| + N \cdot f)) = O(k|\mathcal{R}|) = O(k \cdot N \cdot f)$. In practice, we observed that only a constant number of rules will be in conflict with the *TopRule* in each of the k iteration. Hence the time complexity is expected to be $O(k \cdot N)$.

4 Experiment

In this section, we perform experimental evaluations and analyses.

4.1 Experiment Setup

We use the following algorithms in the experiments:

Greedy This is the state-of-the-art algorithm for learning top-k transformation rules by Microsoft researchers [1].

LA This is our proposed local-alignment-based algorithm.

Note that for our LA algorithm, the weights set for various types of rules in Equation (1) are: rules from mapped tokens: 4, exact acronym: 3, partial acronym: 2, and others: 1. We use a partitioning method to segment each record into three fields based on an algorithm that takes into consideration the local alignment costs. In the interest of space, we will leave it to the full version of the paper.

All experiments were performed on a PC with AMD Opteron Processor 8378 CPU and 96GB memory, running Linux 2.6.32. All programs are implemented in Java.

We use the following three real-world datasets.

CCSB This dataset comes from the Collection of Computer Science Bibiliographies.[2] We queried the site with 38 keyword queries (e.g., `data integration`), and collected the top-200 results. Each query result is referred to a *cluster*, as it may contain multiple citations to the same paper. We only kept those clusters whose size is larger than one. We used five different bibliography styles.[3] We applied the i-th bibliography style to the i-th citations (if any) in each cluster, and got the corresponding transformed string from the LaTeX output. As a result, 3030 clusters were generated. We then form all possible pairs of the strings produced by LaTeX within the same cluster, and use them as the input record pairs for the transformation rule learning algorithms. This results in 12,456 pairs.

Cora This is the hand-labeled Cora dataset from the RIDDLE project.[4] It contains 1,295 citations of 112 Computer Science papers. We clustered the citations according to the actual paper they refer to, and then generated all possible pairs within each cluster. As a result, we had 112 clusters with a total of 17,184 input record pairs.

Restaurant This is the Restaurant dataset from the RIDDLE project. It contains 533 and 331 restaurants assembled from Fodor's and Zagat's restaurant guides, and 112 pairs of coreferent restaurants were identified. We applied similar record pair generation method as above, and generated 112 clusters and a total of 112 record pairs.

Note that we generate all pairs of citations in the same cluster (i.e., referring to the same publication) to exploit the ground truth data maximally. This, however,

[2] `http://liinwww.ira.uka.de/bibliography/Misc/CiteSeer/`

[3] They are `these`, `acm`, `finplain`, `abbrv`, and `naturemag`, mainly from `http://amath.colorado.edu/documentation/LaTeX/reference/faq/bibstyles.pdf`.

[4] `http://www.cs.utexas.edu/users/ml/riddle/data.html`

introduces a bias into the frequency of the rules (i.e., $freq(R)$ in Equation (1)). For example, for a cluster of size $2t$, a rule `A` ⇔ `B` can have a frequency of up to t^2 from this cluster alone. To remedy this problem, we define the frequency as the number of *clusters* (rather than *record pairs*) such that the rule is generated. This is applied to both algorithms.

4.2 Quality of the Rules

For both the Greedy and LA algorithms, we generate top-k rules for each dataset with varying number of k. We then validate *all* the output rules which were classified by domain experts into one of the three categories:

Correct when the rule is absolutely correct. E.g., `proceedings` ⇔ `proc`.
Partially Correct when the rule is partially correct. E.g., `ipl information processing letters` ⇔ `inf process lett vol`.
Incorrect when the rule is absolutely incorrect. E.g., `computer science` ⇔ `volume`.

Some of the correct rules discovered by LA are shown in Table 1.

Table 1. Example Rules Found

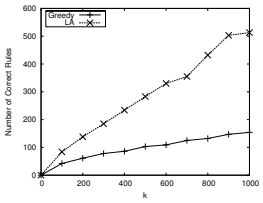
ID	Rule
1	`focs` ⇔ `annual ieee symposium on foundations of computer science`
2	`computer science` ⇔ `comput sci`
3	`pages` ⇔ `pp`
4	`5th` ⇔ `fifth`
5	`maaliskuu` ⇔ `march`

To evaluate the quality of rules generated by the algorithms, we count the number of *correct* and *incorrect* rules. We also calculate *precision* as the fraction of correct rules in the top-k output rules.[5] Note that we essentially ignore partially correct rules.

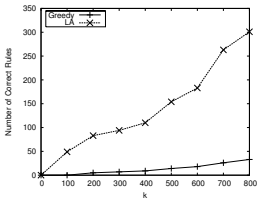
Figures 3(a), 3(b), and 3(c) show the numbers of correct rules for two algorithms by varying k on three datasets. Figures 3(d), 3(e), and 3(f) show the number of incorrect rules by varying k. We can see that

- LA outperforms Greedy substantially on all datasets by generating more correct rules than incorrect rules.
- Since the Cora dataset is dirtier than the CCSB dataset, the precision of both algorithms is lower. It can also been seen that the Greedy algorithm is affected more by the noise in the dataset than the LA algorithm.

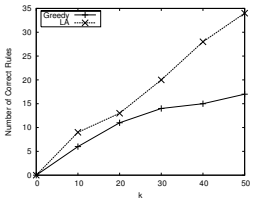
[5] Note that this definition is different from that in [1], where precision was defined as the number of incorrect rules.



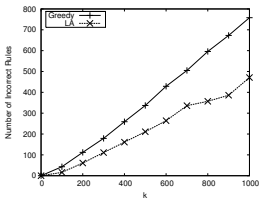
(a) CCSB, Number of Correct Rules



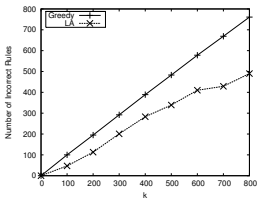
(b) Cora, Number of Correct Rules



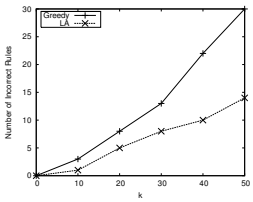
(c) Restaurant, Number of Correct Rules



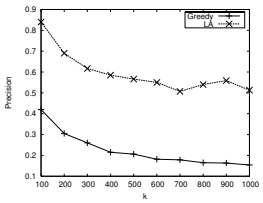
(d) CCSB, Number of Incorrect Rules



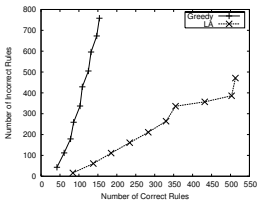
(e) Cora, Number of Incorrect Rules



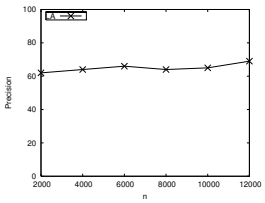
(f) Restaurant, Number of Incorrect Rules



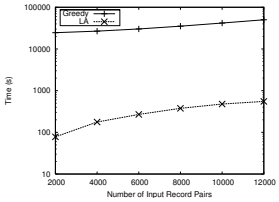
(g) CCSB, Precision vs. k



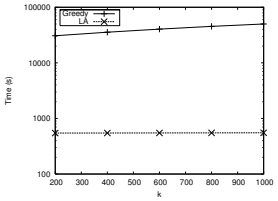
(h) CCSB, Number of Incorrect Rules vs. Number of Correct Rules



(i) CCSB, Precision vs. n



(j) CCSB, Time vs. n ($k = 1000$)



(k) CCSB, Time vs. k ($n = 12,000$)

Fig. 3. Experimental Results

– Since the Restaurant dataset is small, only fewer rules are generated. For the top-50 rules, LA generate 68% correct rules whereas Greedy has a precision of 34%.

We also plot the precision vs. k on the CCSB dataset in Figure 3(g). Since both algorithms strive to find high-quality rules first, the precision is highest when k is small, and decreases with increasing k. Precisions for both algorithms become stable for $k \geq 500$. In all settings, LA's precision is much higher than

Greedy's. As we can see from the figure, the precisions of LA and Greedy is 84% and 42% when $k = 100$, and 51.3% and 15.4% when $k = 1000$.

Figure 3(h) shows how many incorrect rules are generated for a given target of correct rules on the CCSB dataset, as [1] did. It can be seen that much more incorrect rules are generated by the Greedy algorithm than the LA algorithm for any fixed amount of correct rules.

A good transformation rule learner should perform consistently when we vary the number of input record pairs. Such results are shown in Figure 3(i), where the input number of record pairs vary from 2,000 to 12,000, and $k = 200$. It can be observed that the precision of our LA algorithm remains stable (between 62% to 69%).

4.3 Execution Time

We investigate the efficiency of the algorithms versus the number of input record pairs (n). We measure the running time of the algorithms. The result is shown in Figure 3(j). We can see that the time grows more quickly for the Greedy algorithm than the LA algorithm. This is mainly because with the increasing input pairs, there are many more candidate rules generated by the Greedy algorithm as they do not perform careful local alignment. The running time of Greedy is up to 300 times more than LA also because of the vast amount of candidate rules generated.

We plot the running time versus the output size k in Figure 3(k). Both algorithms require more time when k increases, but Greedy's time grows quickly with k. This is mainly because Greedy needs to update the support of the vast amount of candidate rules in each iteration and hence takes much more time.

5 Conclusions

In this paper, we propose an effective and efficient top-k transformation rule learning algorithm. The algorithm is based on performing careful local alignment of input coreferent record pairs, and generating candidate rules based on the optimal local alignment. Our experiments demonstrate that our method generates more correct rules than the global greedy approach [1] in less amount of time.

References

1. Arasu, A., Chaudhuri, S., Kaushik, R.: Learning string transformations from examples. PVLDB **2** (2009) 514–525
2. Winkler, W.E.: The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau (1999)
3. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Trans. Knowl. Data Eng. **19** (2007) 1–16
4. Wang, W., Xiao, C., Lin, X., Zhang, C.: Efficient approximate entity extraction with edit distance constraints. In: SIGMOD. (2009)

5. Winkler, W.E., Thibaudeau, Y.: An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. Decennial Census. Technical report, US Bureau of the Census (1991)
6. Xiao, C., Wang, W., Lin, X., Yu, J.X.: Efficient similarity joins for near duplicate detection. In: WWW. (2008)
7. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string metrics for matching names and records. In: In Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web. (2003)
8. Moreau, E., Yvon, F., Cappé, O.: Robust similarity measures for named entities matching. In: COLING. (2008)
9. Bilenko, M., Mooney, R.J.: Learning to combine trained distance metrics for duplicate detection in databases. Technical report, University of Texas at Austin (2002)
10. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: KDD. (2002) 269–278
11. Tejada, S., Knoblock, C.A., Minton, S.: Learning domain-independent string transformation weights for high accuracy object identification. In: KDD. (2002) 350–359
12. Cohen, W.W., Richman, J.: Learning to match and cluster large high-dimensional data sets for data integration. In: KDD. (2002) 475–480
13. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: KDD. (2003) 39–48
14. Bilenko, M., Mooney, R.J.: On evaluation and training-set construction for duplicate detection. In: Proceedings of the KDD-03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation, Washington, DC (2003) 7–12
15. Minton, S., Nanjo, C., Knoblock, C.A., Michalowski, M., Michelson, M.: A heterogeneous field matching method for record linkage. In: ICDM. (2005) 314–321
16. Arasu, A., Götz, M., Kaushik, R.: On active learning of record matching packages. In: SIGMOD Conference. (2010) 783–794
17. Michelson, M., Knoblock, C.A.: Mining the heterogeneous transformations between data sources to aid record linkage. In: IC-AI. (2009)
18. Arasu, A., Chaudhuri, S., Kaushik, R.: Transformation-based framework for record matching. In: ICDE. (2008) 40–49
19. Arasu, A., Chaudhuri, S., Ganjam, K., Kaushik, R.: Incorporating string transformations in record matching. In: SIGMOD Conference. (2008) 1231–1234
20. Turney, P.D.: Mining the web for synonyms: Pmi-ir versus lsa on toefl. In: ECML. (2001) 491–502
21. Pang, B., Knight, K., Marcu, D.: Syntax-based alignment of multiple translations: Extracting paraphrases and generating new sentences. In: HLT-NAACL. (2003)
22. Bar-Yossef, Z., Keidar, I., Schonfeld, U.: Do not crawl in the dust: different urls with similar text. In: WWW. (2007)
23. Dasgupta, A., Kumar, R., Sasturkar, A.: De-duping urls via rewrite rules. In: KDD. (2008) 186–194
24. Jones, R., Rey, B., Madani, O., Greiner, W.: Generating query substitutions. In: WWW. (2006)
25. Radlinski, F., Broder, A.Z., Ciccolo, P., Gabrilovich, E., Josifovski, V., Riedel, L.: Optimizing relevance and revenue in ad search: a query substitution approach. In: SIGIR. (2008)
26. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: ICML. (2001) 282–289
27. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly **2** (1955) 83–97