Learning String Transformations From Examples

Arvind Arasu
Microsoft Research
One Microsoft Way
Redmond, WA, USA
arvinda@microsoft.com

Surajit Chaudhuri
Microsoft Research
One Microsoft Way
Redmond, WA, USA
surajitc@microsoft.com

Raghav Kaushik
Microsoft Research
One Microsoft Way
Redmond, WA, USA
skaushi@microsoft.com

ABSTRACT

"Robert" and "Bob" refer to the same first name but are textually far apart. Traditional string similarity functions do not allow a flexible way to account for such synonyms, abbreviations and aliases. Recently, string transformations have been proposed as a mechanism to make matching robust to such variations. However, in many domains, identifying an appropriate set of transformations is challenging as the space of possible transformations is large. In this paper, we investigate the problem of leveraging examples of matching strings to learn string transformations. We formulate an optimization problem where we are required to learn a concise set of transformations that explain most of the differences. We propose a greedy approximation algorithm for this NP-hard problem. Our experiments over real-life data illustrate the benefits of our approach.

1. INTRODUCTION

Record matching [7] is the well-known problem of matching records that represent the same real-world entity and is an important step in the data cleaning process. An example of record matching is identifying the customer record in a data warehouse from the customer information such as name and address in a sales record. Due to various reasons such as erroneous data entry and different formatting conventions, the name and address information could be represented differently in the two records, making the task of matching them challenging.

Most approaches to record matching [7] rely on textual similarity of the records, typically computed using a *similarity function* such as *edit distance* and *jaccard similarity*, to determine if two records are matches or not. However, textual similarity can be an imperfect indicator of whether or not two records are matches; in particular, two matching records can be textually dissimilar. A common reason why this happens is alternate representations for the same concept or entity; for example, the first name `Robert` can be written as `Bob`, and `United States of America` can be abbreviated to `USA`.

VLDB '09, August 24-28, 2009, Lyon, France

Prior work [1, 7, 13] has recognized the importance of such representational variations for record matching and has proposed several techniques for incorporating *a priori* knowledge of such variations into record matching process. They have also demonstrated that exploiting variations using these techniques can significantly improve the quality of record matching. Following [1, 7], we use *string transformations* to refer to such alternate representations. We use the notation $x \rightarrow y$ (e.g., `Robert` $\rightarrow$ `Bob`) to denote a transformation.

While previous work [1, 7, 13] describes how a set of transformations can be exploited for record matching, it does not address how to identify suitable transformations in a record matching setting. For a real-world record matching task, hundreds of string transformations could be relevant and it is a challenging task for a programmer to compile the set of relevant transformations.

For example, consider the task of compiling transformations for matching citations such as those shown in Figure 1. A number of transformations are relevant to this matching task. These include conference and journal abbreviations (`VLDB` $\rightarrow$ `Very Large Data Bases`), subject related abbreviations (`Soft` $\rightarrow$ `Software`), date related variations (`Nov` $\rightarrow$ `November`, and `'76` $\rightarrow$ `1976`), number related abbreviations (`8th` $\rightarrow$ `Eighth`), and a large number of variations which do not fall into any particular class (`pp` $\rightarrow$ `pages`, `eds` $\rightarrow$ `editors`). Figure 2 shows a partial list of transformations that are relevant to matching in the domain of organization names. (These transformations were in fact generated using the techniques presented in this paper.) As is apparent, this list is a potpourri of many kinds of transformations such as state abbreviations and school related variations. Manually compiling such a list is a challenging task.

For some popular domains such as US addresses, there are standard pre-compiled sets of transformations [17]. Transformations can also be obtained from online sources—for example, we can use DBLP [5] to compile a set of conference abbreviations. While these are indeed valuable sources, they are rarely comprehensive for a given record matching task. As a concrete example, while the precompiled set of transformations provided by USPS covers variations relating to street name endings (e.g., `Ave` $\rightarrow$ `Avenue`), it does not cover variations relating to street names (e.g., `Univ` $\rightarrow$ `University` and `5th` $\rightarrow$ `Fifth`).

The above scenarios argue the need for other ways of obtaining transformations. In this paper, we consider the approach of *learning* transformations from user-provided ex-

Id	Left	Right
C1	Katayama,T., "A hierarchical and functional software process description and its enaction", Proc. 11th ICSE, IEEE, 1989, pp.343-352	T. Katayama, "A hierarchical and functional software process description and its enaction," In: Proceedings of the Eleventh Int. Conf. On Soft. Eng. Pages: 343–352, IEEE Computer Society Press, Pittsburgh, PA, Jan 1989.
C2	Knuth, D., The art of Computer Programming, Vol. III, Addison-Wesley, (1973).	8. D. Knuth, The art of Computer Programming, Volume 3: Sorting and Searching, Addison-Wesley, Reading, MA, 1973.
C3	[ESWARAN76] Eswaran, K. P., J. N. Gray, R. A. Lorie, I. L. Traiger, "The notions of consistency and predicate locks in a database system", Communications of the ACM, Vol. 19, No. 11, November, 76	[14] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The notions of consistency and predicate locks in a database system," Commun. Assoc. Comput. Mach., Vol. 19, No. 11, Nov. 1976

Figure 1: Matches from citation domain

co	→ company	indep	→ independent	st	→ saint	sd	→ school district
corp	→ corporation	assoc	→ associates	sch	→ school	univ	→ university
inc	→ incorporated	assoc	→ association	cmt	→ community	n w	→ northwest
dist	→ district	ltd	→ limited	tech	→ technology	hosp	→ hospital
ctr	→ center	dept	→ department	PA	→ pennsylvania	intl	→ international
mfg	→ manufacturing	cond	→ conditioning	bros	→ brothers	a c	→ a/c

Figure 2: Example rules relevant to organization names, learned using techniques presented in this paper

amples of matching strings. The idea is that by drawing examples from the data sets being matched, we can identify transformations that are appropriate for the matching task at hand. We argue that such an approach is useful because it is easier for a record matching programmer to provide examples compared to manually compiling lists of transformations. We believe, with the exception of [12], there is very little previous work that addresses this problem. We discuss [12] in more detail later in this section and defer a more detailed discussion of related work to Section 6.

Overview and Contributions

As observed earlier, a transformation introduces textual differences between two matching records, and this observation suggests that we can analyze textual differences between two matching strings (records) to identify transformations. Informally, the *difference* between two strings is parts of one string that is not present in the other, and vice-versa. Consider, for example, the pair of strings E1 from address domain shown in Figure 3. The difference between the two strings is highlighted in bold and corresponds to the transformation `Highway` → `Hwy`.

There are at least two reasons why building on this intuition is challenging: First, more than one transformation could contribute to differences between two matching strings. For example, many transformations such as (`Proc` → `Proceedings`), (`11th` → `Eleventh`), and (`pp` → `Pages`) contribute to the differences between the pair of strings C1 in Figure 1. It is not obvious how we can syntactically analyze the difference between the two strings and attribute them to the "correct" set of transformations above and not to a completely meaningless set of transformations such as (`Proc` → `Proceedings of the Eleventh`) and (`11th` → `Int. Conf.`). Second, not all differences between two matching strings are attributable to meaningful transformations. For example, no meaningful transformation (in the sense of alternate representations) accounts for the presence of the word `Pittsburgh` on the right string of C1 and its absence on the left.

In order to learn meaningful transformations, we analyze a large number of matching strings and seek a *concise* set of *syntactic rules* (candidate transformations) of the form $x \rightarrow y$ that can be used to account for a large part of the differences between each pair of matching strings. Informally, a rule such as `Proc` → `Proceedings` is likely to be in such a concise set, since we expect it to "occur" in a large number of matching strings and so can be used to account for a large part of the differences between matching strings. On the other hand, a rule such as `11th` → `Int. Conf.`, is unlikely to be part of such a concise set since we expect it to occur in few matching strings. In summary, we expect our concise set of syntactic rules to mostly correspond to semantically meaningful transformations.

The *rule learning problem* that we present in Section 2.2 is a formalization of the above ideas and has connections to the *Minimum Description Length (MDL)* principle [9]. We show that the formal learning problem is NP-hard and we provide a simple greedy algorithm in Section 3. The greedy algorithm is an $\frac{1}{2}(1-\frac{1}{e^2})$ approximation under reasonable assumptions, and we demonstrate empirically that it produces useful, meaningful transformations in practice. Further, the greedy algorithm is linear in the input size which allows us to easily scale with the number of examples.

We note that while [12] takes a similar approach of analyzing differences to identify transformations, the candidate rules that they consider are *entire* differences between matching strings. For example, the candidate rule for matches C1 in Figure 1 would be `Proc. 11th ICSE ...` → `In: Proceedings of the Eleventh ....` In other words, [12] does not consider the option of accounting for differences between matches using multiple transformations. This aspect of the problem formulation is essential to identify transformations for complex domains such as citations. We also note that most of the technical complexity of our problem

Id	Left	Right
E1	60460 **Highway** 50 Olathe CO	60460 **Hwy** 50 Olathe CO
E2	60460 **Highway** 50 **(PO Box 2239)** Olathe	60460 **Hwy** 50 Olathe **CO**
E3	599 **N E** 83rd St Redmond **WA**	599 **Northeast** 83rd St Redmond
E4	1932 **Univ** Ave Madison WI	1932 **University** Ave Madison WI

Figure 3: Matches from address domain

formulation arises precisely due to this component.

In Section 4, we discuss how our solution for the transformation learning problem can be leveraged for a given record matching task. In particular, we discuss heuristics for generating large number of example matching strings using limited input from a domain expert, and discuss approaches for validating the learned transformations before deploying them for record matching. We note in this context that the techniques presented in this paper exploit positive examples and not negative examples (i.e., examples of strings that are not matches). Developing techniques that exploit negative examples is an interesting direction of future work.

We conduct a detailed empirical investigation of our algorithm (Section 5) across a variety of real-world data sets. We study various aspects of the algorithm including the quality of output transformations, their impact on the quality of record matching and the impact of the number of input examples. Finally, we discuss related work in Section 6 and conclude in Section 7.

2. PRELIMINARIES

This section provides a brief overview of the record matching problem and then formulates the transformation learning problem, which is the main focus of this paper.

2.1 Record Matching

The record matching problem is the problem of identifying, given two input relations R and S, all pairs of records $(r, s) \in R \times S$ that *match*, i.e., correspond to the same real-world entity. The notion of a match is human-defined and lacks a precise formal characterization.

Current approaches to record matching typically involve a *design phase* where a record matching *program* is developed using input from a domain expert and an *execution phase* where the designed program is evaluated over the inputs R and S [7, 11]. A record matching program can be fairly complex: Apart from the input relations R and S, the program can take as auxiliary inputs a variety of domain knowledge and it can be structured as a complex operator tree of basic record matching primitives such as segmentations and similarity joins. Most of these details are orthogonal to the results of this paper. For the purposes of this paper, we view a record matching program as a black box that can benefit from a set of transformations as an auxiliary input. The details of how transformations are used within a record matching program are also orthogonal to the discussion here.

Our overall goal is to identify transformations relevant to a given record matching problem. Reasoning formally whether or not a transformation is relevant to a given record matching task seems challenging since a record matching program can use a transformation in fairly complicated ways. We instead formulate and study the simpler problem of learning transformations from a given set of example matching strings. In Section 4, we discuss heuristic approaches to use our solution to the simpler problem to identify transformations relevant a given record matching task; the heuristic approaches essentially provide techniques for generating example matching strings.

2.2 Transformation Learning Problem

The input to the transformation learning problem consists of a set of N positive examples, $E^+ = \{\langle X_i, Y_i\rangle : i \in [1, N]\}$, where X_i and Y_i are matching strings from some domain such as addresses or organization names. Our high-level goal is to learn *transformations* from these examples.

We assume all of our strings are sequences of basic units called *tokens*; we typically use words as our tokens. For a string X, $|X|$ denotes the number of tokens in X; $X[i]$ denotes the ith token in X, with $X[1]$ being the first token; $X[i, j]$ $(1 \le i \le j \le |X|)$ denotes the subsequence $X[i] \cdots X[j]$. Whitespace and punctuations serve as delimiters for tokenization and do not themselves form part of a token. For example, if we denote the left string of example E2 in Figure 3 by X, then $|X| = 7$ and $X[4, 6] = \langle$ `PO`, `Box`, `2239` $\rangle$.

We now build on the informal discussion of Section 1 and formalize the transformation learning problem. Consider an example pair $\langle X_i, Y_i\rangle$. Since X_i and Y_i are matching strings, we expect the tokens in X_i to be related to tokens in Y_i. For example, in Example E2 of Figure 3, the tokens `60460`, `50`, `Olathe` on the *Left* are related to identical tokens on the *Right*, and the token `Highway` on the *Left* is related to the token `Hwy` in the right. Figure 4 shows this relation pictorially. As the above example suggests, a simple and common kind of relationship is the *identity* relationship, which arises when the same token is present in both X_i and Y_i. A second kind of relationship arises due to transformations, i.e., when a sequence of tokens in Y_i is an alternate representation of a sequence of tokens in X_i. We now introduce a series of definitions that capture the *syntactic* structure of relating tokens in matching strings. We use these definitions later in our problem formulation.

A *rule* is defined using a pair of strings x and y and is denoted $x \rightarrow y$. The rule $x \rightarrow y$ can be used to relate string x and string y. A rule has the syntactic structure of a transformation, but is different from a transformation. For example, `PO Box 2239` → `CO` is a valid rule, while it would be meaningless as a transformation. To illustrate the use of rules to relate tokens, we can use the rule (`Highway` → `Hwy`) to relate the 2nd token on the left and the 2nd token on the right in Figure 4. As another example, we can use the rule (`Olathe` → `Olathe`) to relate the 7th token on the left and the 4th token on the right. The definition of *rule application* formalizes relating tokens using rules.

Definition 1. *A* rule application *over a pair of strings* $\langle X, Y\rangle$ *is a three tuple* $\langle x \rightarrow y, i, j\rangle$*, where* $x \rightarrow y$ *is a rule*

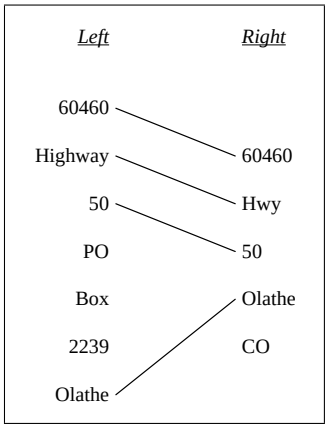


Figure 4: The correct alignment of tokens of the pair E2 of Figure 3

such that $x = X[i, i + |x| - 1]$ *and* $y = Y[j, j + |y| - 1]$. *We say that the above rule application* covers *the tokens* $X[i], \ldots, X[i+|x|-1]$ *and the tokens* $Y[j], \ldots, Y[j+|y|-1]$. □

EXAMPLE 1. *The two rule applications over the strings in Figure 4 we informally described above are* ⟨`Highway` → `Hwy`, 2, 2⟩ *and* ⟨`Olathe` → `Olathe`, 7, 4⟩. *Both of these rule applications cover two tokens. The rule application* ⟨`60460 Highway` → `Olathe CO`, 1, 4⟩ *covers 4 tokens.* □

DEFINITION 2. *An* alignment *of a pair of strings* $\langle X, Y \rangle$ *is a set of non-overlapping rule applications over* $\langle X, Y \rangle$. *Two rule applications over the same pair of strings are* non-overlapping *if the set of tokens they cover do not overlap.* □

EXAMPLE 2. *The alignment corresponding to Figure 4 is:* {⟨`60460` → `60460`, 1, 1⟩, ⟨`Highway` → `Hwy`, 2, 2⟩, ⟨`50` → `50`, 3, 3⟩, ⟨ `Olathe` → `Olathe`, 7, 4⟩. *Another alignment is the set:* {⟨`60460` → `Olathe CO`, 1, 4⟩, ⟨`Highway` → `60460`, 2, 1⟩}. □

An alignment specifies how we can relate two matching strings using a collection of rule applications. We comment on two important aspects of our definition of alignment. First, our definition includes a constraint that two rule applications in an alignment cannot overlap. This is a natural constraint that is typically satisfied by the "correct" way of relating tokens in matching strings (hereafter, *correct* alignment). This constraint is satisfied by the alignment shown in Figure 4 and the reader can verify that this is the case for all of the examples in Figures 1 and 3. This constraint is important for learning high quality transformations as we argue shortly. However, much of the complexity of the transformation learning problem also arises due to this constraint.

Second, our definition of alignment does not impose an ordering constraint over its rule applications. In particular, a valid alignment can contain two rule applications $\langle x \rightarrow y, i, j \rangle$ and $\langle x' \rightarrow y', i', j' \rangle$ where $i < i'$ and $j > j'$ to be part of an alignment. (The second alignment in Example 2 is an instance of such an alignment.) We do not impose the ordering constraint since imposing it might preclude correct alignments. For example, in Example C1 of Figure 1 the token `pp` of the left is related to the token `Pages` of the right, but the token `pp` occurs before the token `IEEE` and token `Pages` occurs after the token `IEEE`, so the correct alignment for this pair of matching strings does not satisfy the ordering constraint. This aspect of our definition of alignment is in the spirit of current approaches to record matching which give less importance to token ordering and use order-independent similarity functions such as cosine and jaccard [7].

Id	Left	Right
1	1st Ave N E	North East First Avenue
2	2nd Ave N	2nd Avenue North
3	3rd Ave E	3rd Avenue East

Figure 5: Sample matching pairs used in Example 3

We use $Align(X, Y)$ to denote the set of possible alignments for a given pair of strings $\langle X, Y \rangle$. The size of $Align(X, Y)$ can be very large; if $|X| = |Y| = n$, then $|Align(X, Y)| = \Omega(2^{n2^n})$, i.e., more than doubly exponential in the sizes of X and Y.

We now present our problem formulation. Using the observation that tokens in matching strings are related by transformations (we can view identity relations as being related by trivial transformations such as `Olathe` → `Olathe`), a natural approach to identifying transformations would be to seek a rule collection $\mathcal{R}$ with the property that every pair of matching strings $\langle X_i, Y_i \rangle \in E^+$ can be completely aligned using only rules in $\mathcal{R}$. A complete alignment of $\langle X_i, Y_i \rangle$ is one that covers every token in X_i and Y_i. Clearly, there are many rule collections with this property and most of them do not contain meaningful transformations. However, we can argue using *Occam's Razor* principle that the *smallest* such rule collection is likely to contain rules that are meaningful transformations. The following example gives intuition why Occam's Razor is a useful heuristic for our problem.

EXAMPLE 3. *Consider the sample matching pairs shown in Figure 5. If we restrict ourselves to small rules of the form* $x \rightarrow y$, $|x| = 1$, $|y| = 1$, *we can verify that the smallest set of rules that completely align all three matching pairs consists of: (*`1st` → `First`*), (*`Ave` → `Avenue`*), (*`N` → `North`*), (*`E` → `East`*), (*`2nd` → `2nd`*), (*`3rd` → `3rd`*). Clearly, these rules correspond to meaningful transformations. The small rules restriction was necessary only because this was a "toy" input with only three matching pairs. Without this restriction, we could have aligned the matching pairs using just three rules, where each rule corresponds to an entire pair (e.g.,* `2nd Ave` → `2nd Avenue North`*).* □

Occam's Razor principle and the related *Minimum Description Length (MDL)* principle [9] are well-known heuristics for the *model selection problem*, which is the problem of picking the best explanation (model) for a set of data observations from among a given set of competing explanations. A well-known example of a model selection problem is the problem of picking the best regular expression that explains (contains) a set of example strings [8]. Informally, Occam's Razor principle says that the best model is the simplest. We can view our problem as a model selection problem where

Id	Left	Right
1	1st Ave N E	North East First Avenue
2	2nd Ave N Apt 5	2nd Avenue North
3	3rd Ave E	3rd Avenue East

Figure 6: Sample matching pairs used in Example 4

the rules (transformations) comprise a model that explains the data (pairs of matching strings).

There are two issues with the problem formulation that seeks the smallest rule collection $\mathcal{R}^*$ that can be used to completely align all input matching pairs. First, even assuming $\mathcal{R}^*$ contains correct transformations, many of them are likely to be trivial identity transformations such as `2nd` → `2nd` in Example 3, which are of little value for record matching. To address this issue, we modify our problem formulation to take as input a set of known transformations (which can include the class of identity transformations and possibly others), and produce as output the smallest rule collection, which along with the known transformations, can be used to completely align the input matching pairs. With this formulation, an algorithm for our problem gets credit only for discovering unknown rules, not for producing trivial identity rules.

Second, the complete alignment requirement in the problem formulation can lead to the inclusion of incorrect rules in $\mathcal{R}^*$, since the correct alignment of a pair of strings is often not a complete alignment (e.g., the alignment in Figure 4). To address this issue, we study the dual version of the problem: For a fixed input parameter k, find a collection of k rules $\mathcal{R}^*$, $|\mathcal{R}^*| = k$, that can be used to align the input strings the most. In other words, we no longer require the alignments to be complete, but we seek to maximize the number of tokens covered by the alignments. The following example illustrates this variant:

EXAMPLE 4. *Consider the sample matching pairs shown in Figure 6. These are identical to those in Figure 5 except for the second pair. To be able to completely align the second pair, we need to include some rule involving* `Apt`*, and any such rule would not be a meaningful transformation. However, in the dual version with $k = 1$, the output is the single rule (*`Ave` → `Avenue`*) which has maximum "coverage" of 6 tokens. As we increase k to 3, the output expands to include the rules (*`E` → `East`*) and (*`N` → `North`*). If we increase k to 4, the output can include an incorrect rule (*`Apt` → `2nd`*) depending on how we break ties.*

Informally, as we increase the value of k, Occam's heuristic suggests that the correct rules (transformations) would appear in the output before the incorrect rules, so by picking a "good" value of k we can identify high quality transformations. For a given input, a good value of k can be determined empirically as we describe in Section 4. We now formalize *coverage* of a rule collection and formally state the transformation learning problem.

DEFINITION 3. *We define the* coverage *of an alignment $\mathcal{A} \in Align(X, Y)$, denoted $Cov(\mathcal{A}, X, Y)$, as the number of tokens of X and Y covered by the rule applications in $\mathcal{A}$. Formally,*

$$Cov(\mathcal{A}, X, Y) \stackrel{\text{def}}{=} \sum_{\langle x \to y, i, j \rangle \in \mathcal{A}} |x| + |y|$$

Given a collection of rules $\mathcal{R}$, the coverage of the collection of rules for a given pair, also denoted $Cov(\mathcal{R}, X, Y)$ is defined as the maximum coverage of an alignment that uses only rules in $\mathcal{R}$. Formally,

$$Cov(\mathcal{R}, X, Y) \stackrel{\text{def}}{=} \max_{\mathcal{A}: \langle r, i, j \rangle \in \mathcal{A} \Rightarrow r \in \mathcal{R}} Cov(\mathcal{A}, X, Y)$$

Finally, the coverage of a collection of rules $\mathcal{R}$ over a given set of input examples E^+, denoted $Cov(\mathcal{R}, E^+)$, is defined as the sum of coverage of $\mathcal{R}$ for each pair of strings in E^+:

$$Cov(\mathcal{R}, E^+) \stackrel{\text{def}}{=} \sum_{\langle X, Y \rangle \in E^+} Cov(\mathcal{R}, X, Y)$$

We define weighted *versions of the coverage definitions above using a* weight function *w that maps tokens to non-negative real values; for a token t, $w(t)$ denotes the weight of the token. In order to define the weighted coverages, we simply define $|x|$ to be the sum of weights of the tokens in x. Unless qualified otherwise, a reference to coverage in the rest of the paper means the unweighted version, not the weighted one.* □

EXAMPLE 5. *Let $\mathcal{R}_I$ denote the collection consisting of all possible single token identity rules, i.e., $\mathcal{R}_I = \{x \to x : |x| = 1\}$. Then the coverage of $\mathcal{R}_I$ for the pair in Figure 4 is 6, which happens to be the coverage of the alignment $\{\langle$*`60460` → `60460`$, 1, 1\rangle$, $\langle$`50` → `50`$, 3, 3\rangle$, $\langle$ `Olathe` → `Olathe`$, 7, 4\rangle$. □

Top-k Rule Learning Problem: Given a *prior* collection of rules $\mathcal{R}_p$, an input set of example matches E^+, and a positive integer k, identify a set of k rules $\mathcal{R}_\delta$, $|\mathcal{R}_\delta| = k$, that maximizes the coverage of $\mathcal{R}_p \cup \mathcal{R}_\delta$ over E^+.

3. RULE LEARNING ALGORITHM

In this section, we present algorithms and hardness results for the top-k rule learning problem introduced in Section 2.2.

The top-k rule learning problem is NP-hard, even for the special case where the prior set of rules $\mathcal{R}_p$ is empty. The problem remain NP-hard even if we restrict ourselves to a simple class of rules called *unit rules*; a unit rule is a rule $x \to y$ with the property $|x| = |y| = 1$. (We call a non-unit rule, a *multi-rule*.)

THEOREM 1. *For a given input set of examples E^+, finding a set of k rules $\mathcal{R}$ that maximizes coverage $Cov(\mathcal{R}, E^+)$ is NP-hard. The problem remains NP-hard even if we restrict ourselves to unit rules.*

Proof: The proof is by reduction from a variant of set-cover. Consider m sets $S_1, \ldots, S_m$ such that $S_i \subseteq \mathcal{U}$ and $\cup_i S_i = \mathcal{U}$ for some universe $\mathcal{U}$. The problem of selecting k sets from among these m sets such that the size of their union is maximized is known to be NP-hard. We construct an instance of the top-k rule learning problem from an instance of this set-cover variant as follows: We map each set S_i to a distinct token t_{si}, and let t_0 be a special token that is not equal to any t_{si}. We construct a pair $\langle X_e, Y_e \rangle$ corresponding to each element $e \in \mathcal{U}$. For each set S_i containing e, X_e

INPUT: Examples $E^+ = \{\langle X_i, Y_i\rangle : i \in [1, N]\}$, rules $\mathcal{R}_p$, and k

1. For $i = 1 \ldots N$
2. $\quad A_i = \text{FINDBESTALIGNMENT}(\mathcal{R}_p, X_i, Y_i)$
3. For $i = 1 \ldots N$
4. $\quad \mathcal{C}_i = \text{GENCANDRULEAPPL}(A_i, X_i, Y_i)$
5. $\mathcal{R}_\delta = \phi$
6. $j = 1 \ldots k$
7. $\quad r = \text{FINDBESTRULE}(\mathcal{C}_1, \ldots, \mathcal{C}_N, A_1, \ldots, A_N)$
8. $\quad$ `// Optional: Validate rule r`
9. $\quad \mathcal{R}_\delta = \mathcal{R}_\delta + r$
10. $\quad$ For $i = 1 \ldots N$
11. $\quad\quad A_i = \text{UPDATEALIGNMENT}(A_i, r)$
12. $\quad\quad \mathcal{C}_i = \mathcal{C}_i - \text{FINDOVERLAPRULEAPPL}(\mathcal{C}_i, \mathcal{A}_i)$
13. Output $\mathcal{R}_\delta$

Figure 7: Greedy algorithm

GENCANDRULEAPPL(A_i, X_i, Y_i):

Output set of all possible rule applications $\langle x \rightarrow y, p, q\rangle$ such that $x = X_i[p, p + l_x]$ and $y = Y_i[q, q + l_y]$ for some $l_x, l_y \geq 0$ and none of the tokens $X_i[p], \ldots, X_i[p + l_x]$, $Y_i[q], \ldots, Y_i[q + l_y]$ are covered by rule applications in A_i.

Figure 8: Generating candidate rules

FINDBESTRULE$(\mathcal{C}_1, \ldots, \mathcal{C}_N, A_1, \ldots, A_N)$:

Return rule r that maximizes $\sum_i Sup(r, A_i, \mathcal{C}_i)$.

Figure 9: Finding best rule

contains the corresponding token t_{si} (in some arbitrary position); Y_e contains a single token t_0. The input to the top-k rule learning problem consists of $E^+ = \{\langle X_e, Y_e\rangle : e \in \mathcal{U}\}$. We can show that if OPT denotes the maximum size of the union of k sets for the instance of the set-cover problem, then the maximum coverage of k unit rules for the instance of the top-k rule learning problem is $2 \cdot \text{OPT}$, and vice-versa, which establishes the NP-hardness. □

We next present a simple greedy algorithm for the top-k rule learning problem. Informally, the greedy algorithm starts with an initial alignment of the input examples using prior rules alone. It then iterates k steps, and at each step picks the "best" rule that can be used to increase the alignment the most. Figure 7 presents a pseudo-code of the greedy algorithm. We first discuss the steps of the greedy algorithm before analyzing its performance.

Finding initial alignment (Steps 1-2*)*: For each $\langle X_i, Y_i\rangle$, the greedy algorithm computes an alignment A_i with maximum coverage (best alignment) using the prior rules $\mathcal{R}_p$ alone. Computing this alignment turns out to be theoretically hard:

THEOREM 2. *For a given collection of rules $\mathcal{R}$ and pair $\langle X, Y\rangle$, computing $Cov(\mathcal{R}, X, Y)$ is NP-hard.*

(Recall that $Cov(\mathcal{R}, X, Y)$ is simply the coverage of the best alignment of the pair $\langle X, Y\rangle$ using rules in $\mathcal{R}$ alone.) Computing the best alignment becomes tractable if all the rules in the collection are unit rules.

THEOREM 3. *If all the rules in a collection $\mathcal{R}$ are unit rules, $Cov(\mathcal{R}, X, Y)$ for a pair $\langle X, Y\rangle$ can be computed in $O(n^{2.376})$ time.*

Proof: The problem of computing $Cov(\mathcal{R}, X, Y)$ can be shown to be equivalent to the problem of maximum bipartite matching, from which the time-complexity follows. □

In record matching applications, the strings X_i and Y_i are typically short token sequence (5-10 tokens). We can use this observation along with the result of Theorem 3 to derive an algorithm for computing the best alignment that works well in practice and handles arbitrary rules (not just unit rules). This algorithm enumerates all possible alignments using multirules alone; for each enumerated alignment, the algorithm extends the alignment by adding unit rule applications to maximize coverage. The latter step can be done efficiently using bipartite matching (Theorem 3). The algorithm outputs the alignment with maximum coverage among all the alignments produced in the above steps. If the number of alignments involving multirules is small, which is the case when $|X_i|$ and $|Y_i|$ are small, the overall algorithm is efficient. Also, note that finding the best alignment is even simpler if all the rules in $\mathcal{R}_p$ are identity rules.

Generate Candidate Rule Applications (Steps 3-4*):* Next, the greedy algorithm considers each alignment A_i computed in Steps 1-2, and generates all possible rule applications $\langle r, p, q\rangle$ that do not overlap with rules applications in A_i. Figure 8 contains the formal specification of the subroutine to generate candidate rule applications. This is exactly the set of rule applications that when added to alignment A_i increase the coverage of A_i. We can show that $r \notin \mathcal{R}_p$, since otherwise A_i would not be the alignment with maximum coverage. The generated rule applications are stored in the variable $\mathcal{C}_i$. The candidate generation algorithm can be suitably changed to allow only rules from a particular class. For example, we could disallow rules with large token sequences, i.e., rules of the form $x \rightarrow y$ where $|x| > L$ or $|y| > L$ for some threshold L.

EXAMPLE 6. *Consider example E1 of Figure 3 and let $\mathcal{R}_p$ denote the set of all identity rules. The best alignment of the pair of strings in E1 involving only identity rules is* {⟨`60460` → `60460`, 1, 1⟩, ⟨`50` → `50`, 3, 3⟩, ⟨ `Olathe` → `Olathe`, 4, 4⟩, ⟨`CO` → `CO`, 5, 5⟩}. *The only candidate rule application for this pair is* ⟨`Highway` → `Hwy`, 2, 2⟩. *Example E2 has 10 candidate rule applications. Some example candidate rule applications are:* ⟨`Highway` → `CO`, 2, 5⟩, ⟨`PO Box 2239` → `CO`, 4, 5⟩, *and* ⟨ `PO Box` → `Hwy` , 4, 2⟩. □

Finding the best rule (Steps 6-7*):* Next, the greedy algorithm iteratively picks k rules. At each step, it picks the rule that increases the coverage of the current alignment $(A_1, \ldots, A_N)$ the most. We define the *support* of a rule $r \in \mathcal{C}_i$ for a given alignment A_i of $\langle X_i, Y_i\rangle$, denoted $Sup(r, \mathcal{C}_i, A_i)$ to be the maximum increase in alignment coverage we can achieve by adding rule applications involving only r. Formally, $Sup(r, \mathcal{C}_i, A_i)$ is defined as $\max_{\Delta r_i}(Cov(A_i \cup \Delta r_i) - Cov(A_i))$, where $A \cup \Delta r_i$ is a valid alignment and Δr_i is a set of rule applications $\in \mathcal{C}_i$ involving only rule r. The subroutine FINDBESTRULE (Figure 9) returns the rule with the maximum support across all input examples.

Update Alignments and Candidate Rule Applications (Steps 10-12*):* After picking each rule r in Step 7, the greedy al-

Id	Left	Right
1	*A B C D*	*A x b C d*
2	*E B F G*	*E F G b*
3	*A B F D*	*A b F d*

Figure 10: Input pairs used in Example 7

i	A_i	$\mathcal{C}_i$
1	$A \to A$, $C \to C$	$B \to x$, $B \to b$, $B \to d$, $B \to x\ b$, $D \to x$, $D \to b$, $D \to d$, $D \to x\ b$
2	$E \to E$, $F \to F$, $G \to G$	$B \to b$
3	$A \to A$, $F \to F$	$B \to b$, $B \to d$, $D \to b$, $D \to d$

Figure 11: Example 7: A_i and $\mathcal{C}_i$ at step 7 for $j = 1$

gorithm updates in Step 11 the current set of alignments $A_1, \ldots, A_N$ by adding the maximum number of rule applications involving r. Note that the increase in the coverage of all these alignments is exactly identical, by definition, to the sum of support of the rule r computed in the previous invocation of FindBestRule. Finally, in Step 12, the algorithm updates the current set of candidate rule applications $\mathcal{C}_1, \ldots, \mathcal{C}_N$ by removing rule applications that overlap with the current alignments $A_1, \ldots, A_N$; these are the rule applications that overlap with the rule applications involving r that were added in the immediately preceding Step 11.

Example 7. *We illustrate the execution of the greedy algorithm for the input examples shown in Figure 10. Let $\mathcal{R}_p$ be the set of all identity rules. Column A_i of Figure 11 shows the best alignment for each input using the identity rules. (Since there is no ambiguity we only show the rules r and not the positions p and q of a rule application $\langle r, p, q\rangle$ in Figures 11 and 12.) Column $\mathcal{C}_i$ of Figure 11 shows the candidate rule applications for these alignments. The best rule in the first iteration of the greedy algorithm ($j = 1$) is $B \to b$ which has a support of 2 for each input example for a total support of 6. Column A_i of Figure 12 shows the updated alignments A_i after adding rule applications involving the rule $B \to b$. Column $\mathcal{C}_i$ of Figure 12 shows the updated candidate rule applications that reflect the addition of the rule applications involving $B \to b$. Note that any rule applications involving B or b has been removed from $\mathcal{C}_i$ in Figure 12 compared to Figure 11. The best rule in the second iteration of the algorithm is $D \to d$ which has a total support of 4. Therefore if $k = 2$, the greedy algorithm produces the two rules $B \to b$ and $D \to d$ as output.* □

Analysis of the Greedy Algorithm

We analyze the performance of the greedy algorithm by comparing the coverage of the rules returned by the algorithm and the rules returned by an optimal algorithm. We begin by considering a special case of the top-k rule learning problem with $\mathcal{R}_p = \phi$ and restricting the class of rules to be just unit rules. For this case, we can prove that the greedy algorithm is an $\frac{1}{2}(1 - \frac{1}{\epsilon^2})$ approximation.

Theorem 4. *Consider an instance of the top-k rule learning problem with $\mathcal{R}_p = \phi$ and let the space of rules be just unit rules. Let $\mathcal{R}_{\text{gdy}}$ and $\mathcal{R}_{\text{opt}}$ respectively denote the set of k rules returned by the greedy algorithm and an optimal algorithm for the above instance. Then, $Cov(\mathcal{R}_{\text{gdy}}) \geq \frac{1}{2}(1 - \frac{1}{\epsilon^2})Cov(\mathcal{R}_{\text{opt}})$.*

i	A_i	$\mathcal{C}_i$
1	$A \to A$, $C \to C$, $B \to b$	$D \to x$, $D \to d$
2	$E \to E$, $F \to F$, $G \to G$, $B \to b$	
3	$A \to A$, $F \to F$, $B \to b$	$D \to d$

Figure 12: Example 7: A_i and $\mathcal{C}_i$ at step 7 for $j = 2$

The proof of the theorem uses ideas from the analysis of the greedy algorithm for the *submodular function optimization problem* [10]; we note however that the coverage function is not submodular. We can show that the greedy algorithm can be implemented in time linear in the input size and the parameter k, assuming that the input strings X_i and Y_i are "small".

Theorem 5. *The running time of the greedy algorithm is $O(N + k)$, where N is the number of input examples in E^+.*

4. INTEGRATION WITH RECORD MATCHING PROCESS

In Sections 2 and 3, we studied the problem of learning transformations from a given set of example matching strings. In this section, we discuss how we can leverage our solution for the transformation learning problem for record matching. We consider two questions: (1) Given a specific record matching problem, how do we identify examples of matching strings that form the input to the transformation learning module? (2) What kind of validation by a domain expert is needed for learned transformations before deploying them for record matching? The techniques that we present in this section are preliminary and heuristic in nature, and we hope to refine them further in future work.

4.1 Source of Examples

A straightforward way of obtaining example pairs of matching records (strings) is to use labeling by a domain expert. However, our approach to learning transformations relies exclusively on positive examples and can benefit significantly from a large number of examples, as our experiments demonstrate (see Section 5). We now discuss a heuristic approach for generating large number of examples using limited input from a domain expert. A similar approach is used in [12].

Since our approach relies on multiple examples, we note that a few *wrong* examples are unlikely to significantly affect the quality of the learned transformations. This observation suggests the following approach to generating a large number of matching examples: We design a traditional record matching program without any use of transformations. Such a record matching program can be a simple approximate string join or a more sophisticated program learned using inputs from a domain expert [3]. We execute this record

matching program over the input tables and use the output matching records (strings) as input to our transformation learning module.

We study this approach empirically in Section 5. We start with a simple record matching program, learn transformations using the output of this program, and use the transformations as an input into the same program. We find that this approach significantly improves the overall record matching recall.

4.2 Validation of Transformations

Since the notion of a transformation is informal (as opposed to the formal notion of a rule), some of the learned transformations may be incorrect. We discuss approaches for handling imprecision in the learned transformations. We study these approaches empirically in Section 5.

The simplest approach is to have a domain expert review the learned transformations and discard the ones marked incorrect. This review can happen in an online or an offline fashion. In the former, a reviewer accepts/rejects rules based on their correctness, in Step 8 of the greedy algorithm (Figure 7). A rejected rule is removed from further consideration and Steps 9-12 are not executed for such a rule. Alternately, the review may be done offline after the algorithm produces its complete output. However, this approach (online or offline) seems feasible when the number of learned transformations is modest, e.g., in the hundreds.

A second approach is to use the learned transformations directly for record matching. While this approach obviates the need for human input for reviewing transformations, incorrect transformations might reduce record matching quality. However, a drop in the precision of record matching can be detected by a domain expert by reviewing a sample of matched records. For the semantics proposed in [1], each pair of matched records has an associated set of transformations that *contribute to* the matching of the records. If a transformation contributes to many incorrect matches and few correct matches, it can be used as evidence that the transformation is incorrect. Exploring other approaches to validation is part of our ongoing work.

5. EMPIRICAL EVALUATION

We now discuss our empirical evaluation of the techniques described in the paper. The goals of our evaluation are as follows.

1. To study the impact of transformation learning on the quality of record matching.
2. To compare the quality of transformation rules generated by our algorithm with alternatives (described later in this section).
3. To study how the number of input examples impacts the quality of transformation rules output.
4. To understand the efficiency of our greedy algorithm as a function of the number of input examples.

5.1 Data

We use two real world data sets for our evaluation. One consists of a set of names and addresses of (organizational) customers of Microsoft Corporation. We call this the `Customer` data set. Each address is segmented into the columns street, city and postal code. The organization name is a single attribute. Each attribute is quite short — the names have an average length of 22 characters and addresses, 16. We have two such feeds of data and the task is to identify pairs that represent the same real-world customer. Each of the tables consists of one million rows.

Data Set	Cardinality	No. of Examples
Customer	1 Million	15,000
Citation	1.3 Million	100,000

Figure 13: Data Sets

The second data set, `Citation` is a collection of citations from a well-known publication database. (The examples in Figure 1 were from this data set.) This table consists of 1.3 million rows. There is only attribute containing a citation string of the kind shown in Figure 1. The record matching task here is to find pairs of strings that refer to the same real world citation. Compared to the customer data set, each attribute here is longer, with an average length of 170 characters.

5.2 Source of Examples

We use the method described in Section 4.1, namely a simple similarity join on the underlying tables to generate examples. The similarity join is configured to achieve 95% output precision. Note that the output could contain multiple columns. We project along h columns (without duplicate elimination) to generate column-specific examples. This translates to 15000 examples for the Customer data set. For the Citation data set, we use a random sample of 100,000 output matches. These specifications are tabulated in Figure 13.

5.3 Algorithms

We first briefly describe some of our implementation details. We tokenize the strings into words and use the weighted version of the greedy algorithm described in Figure 7 with idf (inverse document frequency) weights for words. In order to improve the precision of the transformation rules returned, we use simple filters based on how often the antecedent and precedent of a rule $x \rightarrow y$ co-occur. We refer to our algorithm as `Greedy`.

As noted above, one of the goals of our study is to compare our greedy algorithm with alternative approaches. We consider two alternatives.

One is an approach where we simply return candidate pairs that have a large support. More precisely, we generate candidate rules (until step 4 in Figure 7) and then outputs rules based on support by repeatedly calling the procedure FindBestRule in Figure 7. We refer to this algorithm as `Support`. Note that the main difference from our algorithm is that the current alignment is not modified. Thus, comparing our algorithm to `Support` illustrates the role of alignment.

The other algorithm is based on the previously proposed solution [12] to learn transformations from examples. The approach is similar to `Support` above, except that we use *mutual information* [12] instead of support. We note that the space of candidate transformations considered in [12] is more limited than ours (discussed in Section 1). We however

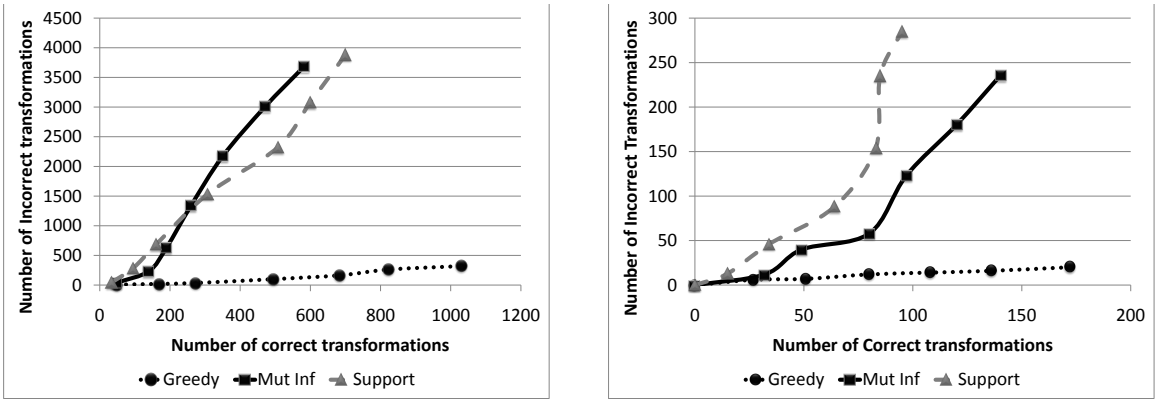
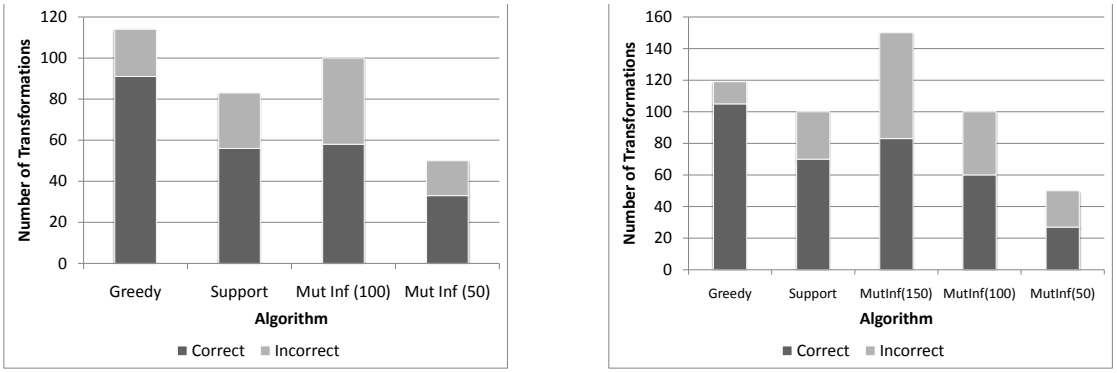


Figure 14: Comparing Rule Learning Algorithms: Citation Data



(*a*) Org Names (*b*) Streets

Figure 15: Comparing Rule Learning Algorithms: Customer Data

use the same space of candidate rules that is considered by `Greedy`. We refer to this algorithm as `MutInf`.

5.4 Comparing Rule Quality

We compare the quality of transformation rules output by the three algorithms described above. We vary the total number of rules generated. At each point, we compute the number of correct rules via human evaluation (we use random sampling when the number of rules is large). Note that the number of correct rules corresponds to *recall* whereas the number of incorrect rules corresponds to *precision*.

We then find how many incorrect rules are generated for a given target of correct rules. We plot this relationship for the three algorithms in Figure 14. The figure on the right is a zoom-in of the figure on the left around the origin. The X-axis shows the target number of correct rules and the Y-axis shows the accompanying number of incorrect rules. Observe that a smaller slope in this graph indicates higher rule quality (fewer incorrect rules for a given number of correct rules).

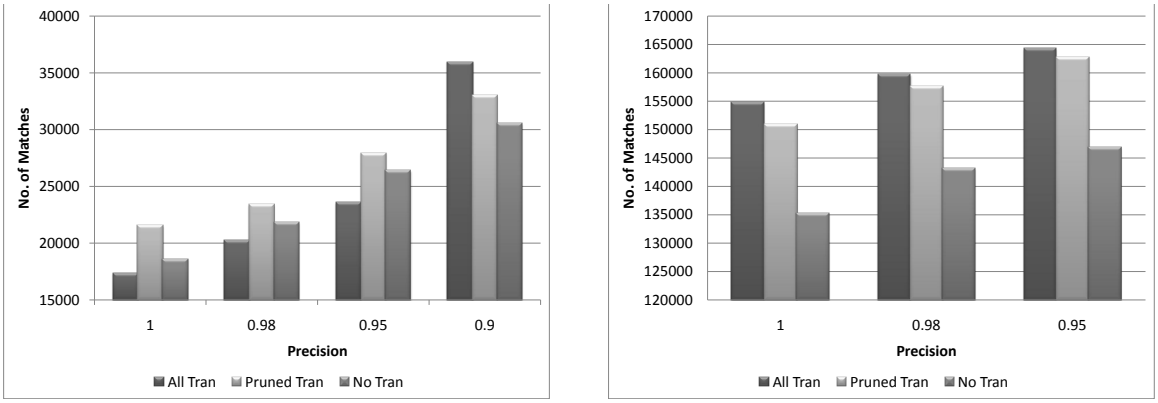
We find that `Greedy` generates over 1000 rules at an overall precision of 76%. In contrast, both `Support` and `MutInf` have significantly poorer quality. In order to generate 500 rules, `Support` yields a precision of 17% and `MutInf` yields a precision of 13%. Thus even though 100s of valid rules are obtained, they require significantly higher validation effort. Observe from the figure on the right that the quality of `Greedy` is the best starting at around 25 rules.

Since the overall number of rules generated is smaller for the Customer data, we analyze the rule quality differently. We generate a minimum number of rules for each algorithm and measure the number of correct and incorrect rules for each algorithm. Figure 15 shows the plot. For `MutInf`, we consider two rule-sets each containing 100 and 50 rules. For both `Support` and `MutInf`, the precision drops off significantly once we increase k beyond what is shown in the figure. Observe that both the precision and recall of `Greedy` is again significantly better. `Greedy` has a precision of 79.8% whereas `Support` has a precision of 67.5% and `MutInf` has precisions of 58% and 66% for $k = 100$ and 50 respectively.

5.5 Impact on Record Matching Quality

We next study the impact of learning transformations on the quality of record matching. The record matching implementation consists of a similarity join where we have the ability to take transformations as input and the underlying similarity function is jaccard similarity [1]. The only record matching input that is varied is the set of transformations. We generate examples by invoking a record matching operation with no transformations at a lower threshold (0.6 in our experiments). These examples are used to generate up to 1000 transformations for each joining column.

We compute the output of three sets of transformations: (1) the empty set ("No Tran"), (2) use all generated transformations without human review ("All Tran"), and (3) perform a human evaluation to prune any incorrect transforma-



(*a*) Customer Data (*b*) Citation Data

Figure 16: Record Matching Quality

tions ("Pruned Tran"). We use the standard quality metrics of *precision* — the fraction of records matched that are correct matches and *recall* measured by the total number of matches returned in the output. Note that the precision can be evaluated by examining a random sample of the output.

We compare the output of the three sets of transformations above as follows. We perform the similarity join (with each rule-set) at various thresholds ranging from 0.6 to 1. We then find the number of matches returned for various values of output precision.

Figure 16 shows the result of our experiments on the Customers and Citation data sets. The X-axis shows the precision and the Y-axis shows the corresponding number of matches found (recall). We note the following points from this Figure.

1. The transformation rules improve the overall record matching quality significantly. For example, at 98% precision, we obtain 7.34% more matches using transformations on the Customers data, and 11.53% more matches on the Citation data. This trend holds for all values of precision in the above Figure.

2. In the Citation data set, using the learned transformations without any human intervention produces the highest recall whereas in the Customers data set, the human intervention helps improve the recall. This shows that the "incorrect" transformations (as defined by human review) do sometimes have the benefit of resulting in more correct matching records. However, in the Customer dataset including all transformations without human review brings down the quality of record matching due to incorrect transformations.

5.6 Input Example Cardinality

As discussed in Section 2.2, our algorithm potentially needs a large number of input examples in order to generate even 100 transformation rules. We empirically investigate this hypothesis over the Citation data set. We consider subsets of the 100,000 example matches of increasing cardinality. For each subset, we find the (largest) number of transformations generated with a precision of 80%. Figure 17 shows the plot where we have the number of input examples on the X-axis and the number of transformations on the Y-axis.

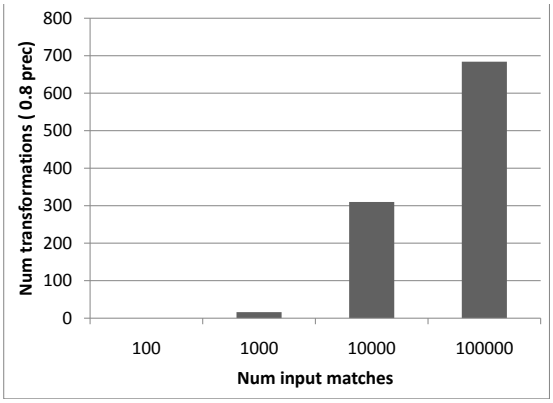


Figure 17: No. of Rules Vs Input Cardinality (Citation Data)

We observe that the number of transformations returned increases sharply. We get virtually no transformations from 100 input examples. On the other hand, the number of transformations returned using 1000 examples is 16 which increases to 310 with 10000 examples.

5.7 Execution Efficiency

Finally, we study the running time of our greedy algorithm. As noted in Section 3, the algorithm is linear which allows it to scale well with the number of examples. We empirically study the performance of our algorithm with increasing number of input examples. We use subsets of increasing cardinality drawn from matches on the Citation data set. Figure 18 shows the running times for various numbers of input examples. We observe a linear increase in running times as the number of input examples grows. This is as expected from our analysis in Section 3.

6. RELATED WORK

Record matching is an active area of research with lot of prior work; see [7, 11] for extensive surveys. A significant portion of research on record matching has focused on designing appropriate similarity functions such as edit distance, jaccard similarity, cosine similarity, and HMM25.

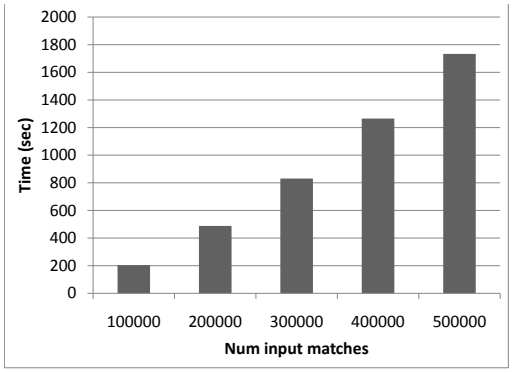


Figure 18: Running Time

A basic limitation of most of them is that they have limited customizability. For example, edit distance can handle typographic errors and jaccard can handle token insertions/deletions, but these similarity functions can not capture other kinds of variations.

In a previous paper [1], we propose a framework for programmable similarity where traditional similarity function is combined with a user-specified set of transformations to derive a most customized similarity function. Other techniques for handling transformations include *standardization*, where input records are preprocessed to change all occurrences of variations (e.g., `US`, `United States`, `USA`) to a canonical form (e.g., `USA`). Traditional record matching techniques are then used over the standardized records. Minton et al [13] propose a machine learning based approach to using transformations, where labeled examples are used to learn which applications of transformations are most useful for matching a pair of records. All of these papers assume that transformations are provided as an explicit input. We believe that [12] is the only prior work that considers the problem of learning transformations, which, as we discussed in Section 1 has several limitations compared to our approach in this paper. Concurrent to this work, Chaudhuri et al [4] consider the problem of determining whether or not two strings are synonymous using information from web pages.

Recent work [6, 15] in record matching has proposed techniques that exploit relationships to learn representational variations. These techniques can reason that if two publications are duplicates and one has a venue `VLDB 2009` and the other `Very Large Data Bases '09`, then these two strings should represent the same concept since each publication has a unique venue. Again, like [12], these techniques cannot "peek" into strings and identify transformations such as `VLDB` → `Very Large Data Bases` and `'09` → `2009`.

Our techniques for learning transformations is a design tool for record matching and therefore this work is related to work on other design tools [3, 14] which learn record matching programs using user-labeled data.

In machine learning literature, Brill and Moore [2] consider the related problem of learning spelling corrections. Given a word such as `detabases`, the goal is to identify corrected words such as `databases`. The overall approach is to learn from example pairs of misspelt and correct words various patterns for misspelling. In a linguistic setting, Turney et al [16] uses word co-occurrence based information retrieval results to identify synonyms.

7. CONCLUSION

In this paper, we studied how we can leverage examples of matching strings to learn string transformations of the form `Bob` → `Robert`. We analyzed the differences between the strings and used the hypothesis that consistent differences occurring across many examples is indicative of a transformation rule. Based on this intuition, we formulated a rule learning problem where we seek a *concise* set of transformation rules that accounts for a large number of differences. We proposed a greedy algorithm to solve this NP-hard problem which yields a factor $\frac{1}{2}(1 - \frac{1}{e^2})$ approximation for an important class of transformation rules, namely unit rules. This algorithm is linear in the input size which allows us to scale easily with the number of input examples. Our empirical study established that this greedy algorithm yields more rules at a higher precision than simpler frequency-based alternatives. We also found a significant impact on record matching quality.

Several questions remain open. It is not clear what is the level of human intervention necessary with the transformations learned. Sometimes it helps improve the record matching quality as expected but sometimes it does not. This happens because transformation rules that are deemed incorrect by human evaluation do result in improving the record matching quality. Another question that remains unanswered is how this transformation learning tool will interact with other record matching design tools that help choose the similarity functions and set appropriate thresholds. We hope to address these questions in future work.

8. REFERENCES

[1] A. Arasu, S. Chaudhuri, and R. Kaushik. Transformation-based framework for record matching. In *Proc. of the 24th Intl. Conf. on Data Engineering (ICDE)*, pages 40–49, Apr. 2008.

[2] E. Brill and R. C. Moore. An improved error model for noisy channel spelling correction. In *Proc. of the 38th Ann. Meeting of Assoc. for Comp. Linguistics (ACL)*, Oct. 2000.

[3] S. Chaudhuri, B. C. Chen, V. Ganti, and R. Kaushik. Example-driven design of efficient record matching queries. In *Proc. of the 33rd Intl. Conf. on Very Large Data Bases (VLDB)*, pages 327–338, Sept. 2007.

[4] S. Chaudhuri, V. Ganti, and D. Xin. Mining document collections to facilitate accurate approximate entity match, 2009. Under submission.

[5] DBLP. `http://www.informatik.uni-trier.de/~ley/db/index.html`.

[6] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proc. of the 2005 ACM SIGMOD Intl. Conf. on Management of Data*, pages 85–96, June 2005.

[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowledge and Data Engg.*, 19(1):1–16, Jan. 2007.

[8] M. N. Garofalakis, A. Gionis, R. Rastogi, et al. XTRACT: A system for extracting document type descriptors from XML documents. In *Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data*, pages 165–176, May 2000.

[9] P. D. Grunwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[10] D. Kempe, J. M. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proc. of the 9th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 137–146, Aug. 2003.

[11] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *Proc. of the 2006 ACM SIGMOD Intl. Conf. on Management of Data*, pages 802–803, June 2006.

[12] M. Michelson and C. A. Knoblock. Mining heterogeneous transformations for record linkage. In *Proc. of the 6th Intl. Workshop on Information Integration on the Web (IIWeb)*, pages 68–73, 2007.

[13] S. Minton, C. Nanjo, C. A. Knoblock, et al. A heterogeneous field matching method for record linkage. In *Proc. of the 5th IEEE Intl. Conf. on Data Mining (ICDM)*, pages 314–321, Nov. 2005.

[14] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of the 8th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 269–278, July 2002.

[15] P. Singla and P. Domingos. Collective object identification. In *Proc. of the 19th Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1636–1637, Aug. 2005.

[16] P. D. Turney. Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. *Lecture Notes in Computer Science*, (2167):491–503, 2001.

[17] United States Postal Service (USPS). `http://www.usps.com`.