Effective Parameter-free Boolean Instance Matching

ABSTRACT

Instance matching is an important step in data integration where the goal is to find instance representations referring to the same real-world thing. State-of-the-art methods use training data to learn combinations of attributes, similarity functions and thresholds, called instance matching rules, for finding matches. The learning of complex rules with thresholds is however complex and thus, very sensitive to training data and parameters. In this paper, we explore a different avenue, proposing an approach that does not use thresholds but more simple *boolean* similarity functions. We show that the simple boolean nature of the employed rules allows for a *parameter-free* learning approach. For high effectiveness, we propose to incorporate fine-grained word-level evidences into rule learning. That is, instead of capturing the similarity of entire attribute values in the rules, our approach employes words extracted from attribute values. Using benchmark matching tasks, we show the proposed solution greatly outperforms state-of-the-art approaches in terms of result quality and most importantly, is not sensitive to the choice of training data and parameters.

1. INTRODUCTION

Effective methods for data integration are crucial for dealing with the large and increasing amount of structured data and data sources. One challenge in data integration is to reconcile differences in the structured data representation of the same real-world thing. It is studied as the problem of *instance matching* (also referred to as entity resolution, entity co-reference, record linkage etc.), which is about finding structured data descriptions of objects (henceforth, simply called instances), which denote the same real-word thing. For example, several product instances collected from different sources vary in their syntactic descriptions (e.g. differences in `Price`, `ID`, etc.) but denote the same thing, the `Apple Macbook Air`.

State-of-the-art approaches employ instance-matching rules to solve this problem [5, 14]. For instance, with the rule "two product instances are the same if they have similar values for `Title` and `Manufacture`" for the data in Tab. 1, the instances n_3 and n_4 can be considered as the same. However, a big challenge is actually to determine how similar is similar [14]. To address this, exiting approaches incorporate *thresholds* into the rules so that instances and their attribute values are only considered similar if their similarity is higher than a threshold. Using different thresholds for different combinations of attributes and similarity functions greatly improves the quality of instance matching. However, the search space for learning these parameters is very complex. As a result, state-of-the-art approaches based on the use of thresholds are sensitive to training data and always require some manual efforts in parameter tuning to be applicable in practice.

In this paper, we explore a different avenue, proposing an effective parameter-free instance matching approach that relies only on relatively simple *boolean similarity functions*. Existing approaches use thresholds to obtain more "sophisticated" evidences because they compare instances at the more coarse-grained level of attribute values. As an example, one evidence might be "the similarity on `Title` for the two instances n_i and n_j is greater than 0.9". Our boolean approach extracts evidences at the more fine-grained level of words (tokens in general) found in attribute values. From the words `13.3`, `Apple`, `ASUS` etc. found in the `Title` values, our approach learns what we call *word-level dissimilarity evidences*, such as "`Apple` and `ASUS` are dissimilar words, one is in the `Title` of n_i and the other is in the `Title` of n_j". It is a dissimilarity evidence because it leads to the inference that n_i and n_j are not the same (are non-matches). We show that using this type of evidences has the following merits: (1) due to their boolean nature, the learning of these evidences is more simple, i.e. does not require parameters and is not sensitive to training data. (2) At the word level, a large number of these evidences can be learned to identify non-matches (high recall). (3) Also due to the use of fine-grained words, the learned evidences are more discriminative in identifying non-matches (high precision).

Using benchmark matching tasks, we show our boolean solution greatly outperforms state-of-the-art approaches in terms of result quality and is superior in terms of sensitivity to training data and parameters. While our focus is to explore the direction of boolean matching, we also discuss in the paper how our boolean approach can be combined with existing works that use thresholds. In the experiment, we show that when used as a boolean filtering mechanism, our approach consistently improves the results of the underlying

matching approach.

Outline We provide an overview in Sec. 2. The learning of dissimilarity evidences is presented in Sec. 3. An implementation of our approach is discussed in Sec. 4. We present experiments in Sec. 5, related works in Sec. 6 and conclusions in Sec. 7.

2. INSTANCE MATCHING

We consider the kinds of data where instances can be represented as sets of attribute values. For the purpose of our work, tokens in attribute values are called words and every value is also conceived as a bag of words. A summary of the main notations used in this paper is shown in Tab. 2

DEFINITION 1 (DATA). *The* data $N[a_1, a_2, ..., a_m]$ *captures a set of instances. Each* instance $n \in N$ *is represented by a set of attributes* $a_i, i \in [1, m]$. *The* value *of an* attribute a_i *of the instance* n *is denoted by* $n[a_i]$. *When conceiving a value* $n[a_i]$ *(or a set of values* $N[a_i]$ *containing* $n[a_i]$ *for each* $n \in N$*) as a bag of words, we use* $W_{n[a_i]}$ *(*$W_{N[a_i]}$*).*

The problem tackled in this work is *instance matching*. Given a set of instances N, the goal is to detect those that refer to the same real-world object.

Table 2: A summary of notations used in this paper.

Notation	Description
N	a set of instances
a	an attribute
$\mathbf{M}^+$, M^+, M'^+, M_i^+	all matches, examples, self-matches, matches in M_i
$\mathbf{C}$, C, C', C_i	Cart. product of words in $\mathbf{M}^+$, M^+, M'^+, M_i^+
$\mathbf{C}^+$, C^+, C'^+, C_i^+	word co-occurrences in $\mathbf{M}^+$, M^+, M'^+, M_i^+
$\mathbf{C}^-$, C^-, C'^-, C_i^-	CDEs for $\mathbf{M}^+$, M^+, M'^+, M_i^+
$W_{N[a]}$ $(W_{n[a]})$	words in a value of all instances (of instance n)

2.1 Thresholded Instance Matching

Existing approaches utilize similarity functions for instance matching, which map a pair of attribute values $(n[a], n'[a])$ to a similarity score in $[0, 1]$. We explicitly distinguish between these two types of similarity functions:

DEFINITION 2 (SIMILARITY FUNCTION). *A* boolean function $same : N[a] \times N[a] \rightarrow \{0, 1\}$ *maps a pair of attribute values to the score* 0 *or* 1. *A* thresholded similarity function $sim : N[a] \times N[a] \rightarrow [0, 1]$ *maps a pair of attribute values to a score in the range* $[0, 1]$.

Based on that, instance-matching rules are used to compute whether a pair of instances (n, n') forms a match:

DEFINITION 3 (INSTANCE-MATCHING RULE). *An* instance-matching rule Λ *is a conjunction of boolean predicates, i.e.* $\Lambda = \bigwedge \lambda_i$, *where* $\lambda_i : [0, 1] \rightarrow \{true, false\}$. *Each* λ_i *is either based on*

- *a thresholded similarity function* sim_j *such that* $\lambda_i(sim_j(n[a_i], n'[a_i]), \theta_{i,j})$ *returns true iff* $sim_j(n[a_i], n'[a_i]) \geq \theta_{i,j}$ *and* $false$ *otherwise,*
- *or a boolean function* $same_j$ *such that* $\lambda_i(same_j(n[a_i], n'[a_i]))$ *returns true iff* $same(n[a_i], n'[a_i]) = 1$ *and* $false$ *otherwise.*

A pair of instances are consider as the same, called a match, *if it satisfies all the boolean predicates in the rule.*

EXAMPLE 1. *An instance matching rule based on thresholded similarity functions is* $\Lambda = \lambda_1(Jaccard(\texttt{Title}), 0.6) \bigwedge \lambda_2(Cosine(\texttt{Manufacture}), 0.5)$. *It indicates that two products are the same if the Jaccard similarity for their* `Title` *value is greater than 0.6 and the Cosine similarity for their* `Manufacture` *value is greater than 0.5.*

State-of-the-art approaches employ thresholded similarity functions [5, 8, 14, 17]. Their learning algorithms rely on training examples to learn (1) the attributes a_i and for each of them, (2) the similarity functions sim_j and for each function, (3) the threshold $\theta_{i,j}$. Compared to the boolean outputs $\{0, 1\}$, scores in the $[0, 1]$ range produced by these approaches are more fine-grained and thus, help to further distinguish the quality of the matching candidates. However, we observe the following drawbacks:

Sensitivity to Training Data. Threshold candidates are generated from the similarities observed in the examples. Intuitively, the goal is to find a threshold that is higher than all similarity scores computed for negative examples and smaller than all scores computed for positive examples. The threshold learned this way is not only sensitive to differences in the training data sample provided as input but also the mix of positive and negative examples. Besides, existing approaches require at least some of the other parameters, i.e. the number of rules, the attributes and the similarity functions, to be selected and tuned manually. The learned thresholds are also sensitive to differences in the choice of these parameters.

Unstable Performance. Often, there is no optimal threshold that can perfectly separates positive from negative examples. Using Jaccard similarity and the data shown in Tab. 1 for instance, we obtain for `Title` the similarity scores 0.19 and 0.42 for (n_1, n_2) and (n_1, n_5), respectively. However, (n_1, n_2), which has a lower score, is actually correct while the other is incorrect. A threshold that identifies all candidates with a score higher than 0.19 as matches can deal with this case. However, it is too low for many other cases, i.e. would return many non-matches. In fact, many thresholds may exist that largely vary but are equally optimal w.r.t. the objective function used for learning. Each of them can provide good results for some cases but does not generalize over all cases. In other words, it is hard to find a threshold that provides stable performance.

2.2 Boolean Instance Matching.

We explore the use of simple boolean functions that are more easy in that learning does not require or is less sensitive to training data. In order to address the second drawback, i.e. to achieve good and stable performance, we employ two strategies. (1) Leveraging the more fine-grained level of words that are contained in attribute values, a large quantity of evidences can be learned. (2) Further, not only the initial set of training examples is used. We propose methods for training data enrichment such that in the end, the entire dataset is employed to derive evidences. In fact, we show that with these methods, our approach also performs well when only automatically computed pseudo-examples are used.

Two types of boolean functions are used in our approach. The first computes a similarity score based on boolean evidences for matching, i.e. *similarity evidences*. Examples for boolean similarity evidences used in literature are value

Table 1: A sample of product instances taken from a real E-commerce database; matching instance pairs are (n_1, n_2), (n_3, n_4), (n_5, n_6), (n_5, n_7), (n_6, n_7).

ID	Title	Manufacture
n_1	Apple MacBook Air 33,8 cm (13,3 inches) Notebook (Intel Core i5, 1,8GHz, 4GB RAM, 128GB hdd, Intel HD 4000)	Apple
n_2	MacBook Air MD231D laptop Core i5-13,3 inches	Apple Inc.
n_3	MacBook Pro with Retina Display 33.8 cm (13.3 inches) laptop Intel Core i7, 2.4GHz, 8GB RAM, 256G SSD, NVIDIA GeForce GT 650M, Mac OS	Apple Inc.
n_4	Apple MacBook Pro ME664LL Notebook with Retina Display (Intel Core i7, 2.4GHz, 8GB RAM, 256G hard disk, NVIDIA GeForce GT 650M)	Apple
n_5	Asus Prime Laptop UX31A-R4003V 33.8 cm (13.3 inches) Ultrabook (Intel Core i7-3517U, 1.9 GHz, 4GB RAM, 256GB HDD, Intel HD 4000)	ASUSTeK Computer Inc.
n_6	ASUS UX31A R4003V Notebook - Core i7 1.9 GHz - 13.3 inch - 4 GB RAM - 256 GB HDD	ASUS
n_7	ASUS Laptop Core i7 1.9 GHz, 13.3 inch, 4 GB RAM, 256 GB HDD	ASUS
n_8	ASUS Core i7 notebook, with 15 inch, 4 GB RAM, 256 GB SSD	ASUS

overlap or substring match. We use this type to quickly find match candidates, a commonly employed pre-processing step also referred to as blocking or candidate selection [17, 10]:

Definition 4 (Candidate Selection). *Given all possible pairs of instances, $(n, n') \in N \times N$,* candidate selection *based on a given attribute a returns a subset $M \subseteq N \times N$ called* matching candidate set*, which is computed using the boolean function same, i.e. $M = \{(n, n') \in N \times N | same(n[a], n'[a]) = 1\}$.*

After candidate selection, we apply another type of boolean functions to the resulting candidates to filter out those that are non-matches. Since they are based on evidences for non-matching, i.e. *dissimilarity evidences*, we refer to this type as *dissimilarity function*. As an example, the fact that two given publications are published in different years can be taken as a dissimilarity evidence to infer that they are non-matches. Applying this type of functions to the candidate set is referred to as the *filtering* step:

Definition 5 (Filtering). *Given a matching candidate set M,* candidate filtering *returns a subset $M^+ = M \setminus M^-$, where M^- represents non-matching candidates computed by the dissimilarity function $\neg same$, i.e. $M^- = \{(n, n') \in M | \neg same(n[a_i], n'[a_i]) = 1\}$.*

We use an existing candidate selection method that uses boolean similarity evidences to quickly produce matching candidates [2]. The focus of this work lies in filtering these candidates based on fine-grained dissimilarity evidences captured at the level of words. Considering attribute values as a whole, few or no dissimilarity evidences can be derived. As an example, while it makes intuitive sense, values for `Manufacture` actually cannot be used as dissimilarity evidences. The dissimilarity function based on that would miss the correct match (n_1, n_2) in Tab. 1 because n_1 and n_2 have different values for `Manufacture`. However, specific words in the values can serve as evidences. For instance, we define that `Apple` and `ASUS` form a dissimilarity evidence, while `Apple` and `Inc.` do not. Thus, given two instances with values for `Manufacture` that contain `Apple` and `ASUS` as words, they will be correctly filtered as a non-match, while the match (n_1, n_2) is preserved because `Apple` and `Inc.` do not represent a dissimilarity evidence.

3. LEARNING WORD-LEVEL DISSIMILARITY EVIDENCES

In this section, we first introduce evidences that are based on word-level co-occurrences. Then, we discuss the problems of learning them from (the possible lack of) training data and finally, our solution for these learning problems.

3.1 Word Co-occurrence Based Evidences

We find dissimilarity evidences in the form of word co-occurrences, namely those pairs of words that when observed in a instance pair, render it as a non-match. We capture this notion of word co-occurrences as follow:

Definition 6 (Word Co-occurrence). *Given an attribute value pair $(n[a], n'[a])$ and their bags of words $W_{n[a]}$ and $W_{n'[a]}$, a word pair $(w_i, w_j) \in W_{n[a]} \times W_{n'[a]}$ co-occurs in $(n[a], n'[a])$, denoted by $(w_i, w_j) \in (n[a], n'[a])$, if $w_i \in W_{n[a]}$ and $w_j \in W_{n'[a]}$, or $w_i \in W_{n'[a]}$ and $w_j \in W_{n[a]}$. Let M be a set of instance pairs. For a given attribute a, the word pair (w_i, w_j) is said to co-occur in M, $(w_i, w_j) \in M$, if there exists an instance pair $(n, n') \in M$ such that $(w_i, w_j) \in (n[a], n'[a])$. We also use $(w_i, w_j) \notin (n[a], n'[a])$ ($(w_i, w_j) \notin M$) to denote that (w_i, w_j) does not co-occur in $(n[a], n'[a])$ (in M).*

Now we introduce the dissimilarity function that uses word pairs as dissimilarity evidences to filter non-matches:

Definition 7 (Diss. Evidence/Function). *Let $C = \{(w_i, w_j) \in W_{N[a]} \times W_{N[a]} | w_i \neq w_j\}$ be all possible word pairs in the values of the attribute a, i.e. values of a for all instances N (denoted by $N[a]$). $C^- \in C$ is a set of word pairs representing dissimilarity evidences, called* co-occurrence based dissimilarity evidences *(CDE), such that given C^-, the function $\neg same : N[a] \times N[a] \rightarrow \{0, 1\}$, called* CDE-based dissimilarity function*, maps a pair of attribute value $(n[a], n'[a])$ to 1, if there exists a word pair $(w_i, w_j) \in C^-$ that co-occurs in $(n[a], n'[a])$, i.e. $(w_i, w_j) \in (n[a], n'[a])$, and 0 otherwise.*

Intuitively, this dissimilarity function determines a pair of values $(n[a], n'[a])$ to be a non-match when they contain a dissimilarity evidence (a word pair $(w_i, w_j) \in C^-$).

3.2 Learning CDE from Positive Examples

The CDE and functions introduced above are more easy to learn because they require only positive examples. This is because by Def. 7, positive examples representing matches must not contain any CDEs. In other words, CDEs cannot co-occur in positive examples. Further, we can even show

that all words pairs that do not co-occur in positive examples must be CDEs such that for computing them, we only need to (1) find all word pairs co-occurring in positive examples, then (2) derive its complement set containing all those word pairs that do not co-occur in positive examples. That complement set must contain all CDEs. We establish the following theorem to enable the learning of CDEs:

THEOREM 1. *Let* $\mathbf{M}^+$ *be the set of* all *matches,* $\mathbf{C}^+$ *be all word pairs that co-occur in* $\mathbf{M}^+$ *and* $\mathbf{C}^-$ *the set of all CDEs. If a word pair* (w_i, w_j) *is not in* $\mathbf{C}^+$*, it is a CDE, i.e.* $(w_i, w_j) \notin \mathbf{C}^+ \Rightarrow (w_i, w_j) \in \mathbf{C}^-$.

PROOF. Given the word pair $(w_i, w_j) \notin \mathbf{C}^+$, we firstly show that all instance pairs containing (w_i, w_j) must be non-matches. Let $M = \{(n, n') | (w_i, w_j) \in (n[a], n'[a])\}$ be all the pairs of instances in which (w_i, w_j) co-occur. We prove $M \cap \mathbf{M}^+ = \emptyset$ by contradiction: assuming M contains some matches, i.e. $M \cap \mathbf{M}^+ \neq \emptyset$. Then, because $\mathbf{C}^+$ contains all the word pairs that co-occur in $\mathbf{M}^+$, $(w_i, w_j) \in (n, n')$ for all $(n, n') \in M \cap \mathbf{M}^+$, then $\mathbf{C}^+$ must contain (w_i, w_j), i.e. $(w_i, w_j) \in \mathbf{C}^+$. This however, cannot be satisfied because as given, $(w_i, w_j) \notin \mathbf{C}^+$.

Because all the instance pairs containing (w_i, w_j) are non-matches, i.e. $M \cap \mathbf{M}^+ = \emptyset$, (w_i, w_j) is a CDE and thus $(w_i, w_j) \in \mathbf{C}^-$. □

Previously, we use $\mathbf{M}^+$ to denote the set of all matches. It is used to define the concept of CDE and shall be computed through instance matching. When training examples are given, we have a subset of all matches. To facilitate the following discussion, we introduce M^+ to refer to such a subset of matches ($M^+ \subset \mathbf{M}^+$). Further, W_{M^+} denotes the bag of words model of the values of instances in M^+, $C = W_{M^+} \times W_{M^+}$ is the Cartesian product representing all possible pairs of words in W_{M^+}, $C^+ \in C$ denotes all word pairs that co-occur in M^+ and $C^- \in C$ are word pairs not co-occurring in M^+.

We employ a word pair graph (WPG) as an intuitive way to capture word pairs in positive examples and its complement set. Intuitively, two words co-occur in positive examples if they are connected by an edge in the WPG, and they do not if there are no edges connecting them. Then, a WPG can be defined as:

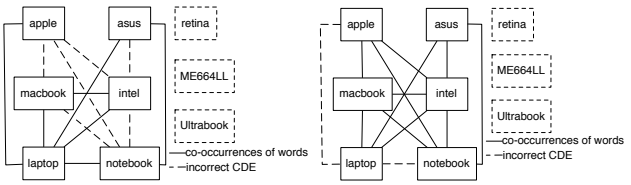
DEFINITION 8 (WORD PAIR GRAPH). *The* word pair graph *is an undirected graph* $G = (V, E)$*. Every node* $v \in V$ *stands for a word in* W_{M^+}*, and every edge* $e(v_i, v_j) \in E$ *captures the co-occurrence between two words* w_i *and* w_j *such that there is an edge* $e(v_i, v_j) \in E$ *iff there is a word pair* $(w_i, w_j) \in C^+$.

Clearly, when the set $\mathbf{M}^+$ of all matches is given, we can then compute a complete WCG that captures the set $\mathbf{C}^+$ of all word pairs that co-occur in $\mathbf{M}^+$, and finally, derive the set $\mathbf{C}^-$ of all and correct CDEs from "missing edges" in that complete WCG according to Theorem 1. However, we only have a subset M^+ of positive examples for CDE learning. Based on Theorem 1, we compute CDEs as

$$C^- = C \setminus C^+ \tag{1}$$

The solution C^- derived from an incomplete set M^+ (incomplete C and C^+) might be incorrect and incomplete.

Incorrect CDE. When M^+ does not contain all matches, then some word pairs in $\mathbf{C}^+$ might not be contained in C^+.



(a) WCG derived from positive examples. (b) WCG derived from self-matches.

Figure 1: Solid lines indicate word pairs co-occurring in examples. Missing lines between any two nodes capture correct CDEs and dotted lines represent incorrect CDEs. Dotted squares indicate words that are not in the examples.

Because CDEs are derived as the complement set of C^+, these word pairs would be included in the solution C^-. However, they actually co-occur in matches (i.e. they co-occur in $\mathbf{C}^+$) and thus shall not be used as CDEs.

Incomplete CDE. The solution is incomplete when there are word pairs in $\mathbf{C}$ that are not in C, i.e. the values in $\mathbf{M}^+$ contain more unique words than the values in M^+. CDEs that belongs to $\mathbf{C} \setminus C$ cannot be derived.

EXAMPLE 2. *Consider the WCG in Fig. 1(a) that is computed based on* $M^+ = \{(n_1, n_2), (n_6, n_7)\}$*. The solid lines indicates word pairs co-occurring in* M^+*, such as* (`asus`, `laptop`) *and* (`laptop`, `notebook`)*. All the missing edges are then derived as CDEs, such as* (`apple`, `asus`) *and* (`macbook`, `asus`)*.* M^+ *is incomplete because it does not include the matches* (n_3, n_4)*,* (n_5, n_6) *and* (n_5, n_7)*. As a result,* (`macbook`, `notebook`) *and* (`asus`, `intel`) *for instance (indicated by the dotted lines), are derived as CDEs because that do not co-occur in* M^+*. However, they co-occur in* (n_3, n_4) *and* (n_5, n_6)*, which are matches not included in* M^+*. Further, correct CDEs such as* (`ME664LL`, `Ultrabook`) *and* (`asus`, `retina`) *for instance, cannot be derived because* `retina`*,* `ME664LL` *and* `Ultrabook` *are not in* M^+.

The goal of this work is to derive a solution C^- that contains all and only correct CDEs from positive examples.

DEFINITION 9 (COMPLETENESS AND SOUNDNESS). *Let* $\mathbf{C}^-$ *be the ground truth that contains all CDEs and* C^- *the learned solution, the learning goals are to minimize* $\mathbf{C}^- \setminus C^-$ *(*$\mathbf{C}^- \setminus C^- = \emptyset$ *means the solution is* complete*) and to minimize* $C^- \setminus \mathbf{C}^-$ *(*$C^- \setminus \mathbf{C}^- = \emptyset$ *means all CDEs in the solution are* sound*).*

Note that $\mathbf{C}^- \setminus C^-$ and $C^- \setminus \mathbf{C}^-$ stand for false negatives and false positives, respectively. Using false positives (incorrect CDEs) for filtering, instance pairs are pruned that are actually correct. This would result in a decrease in *recall*. On the other hand, when filtering is based on a solution with false negatives (an incomplete set of CDEs), some non-matches cannot be pruned, resulting in a decrease in *precision*. By maximizing the completeness and soundness of CDE learning, we aim to maximize the recall and precision of instance matching.

We provide the following theorems to reduce the false positives and the false negatives:

THEOREM 2. *The set of false positives* $C^- \setminus \mathbf{C}^-$ *equals* $(\mathbf{C}^+ \cap C) \setminus C^+$.

PROOF. Because $C^- \subseteq C$, we have $C^- \setminus (\mathbf{C}^- \setminus C) = C^-$. Then,

$$\begin{aligned} C^- \setminus \mathbf{C}^- &= C^- \setminus ((\mathbf{C}^- \cap C) \cup (\mathbf{C}^- \setminus C)) \quad (2)\\ &= C^- \setminus (\mathbf{C}^- \setminus C) \setminus (\mathbf{C}^- \cap C)\\ &= C^- \setminus (\mathbf{C}^- \cap C). \end{aligned}$$

Because $\mathbf{C}^- \cap C^+ = \emptyset$, we can write $\mathbf{C}^- \cap C = (\mathbf{C}^- \cap C) \setminus C^+$. Then,

$$\begin{aligned} C^- &= C \setminus C^+\\ &= ((\mathbf{C}^+ \cap C) \cup (\mathbf{C}^- \cap C)) \setminus C^+ \quad (3)\\ &= ((\mathbf{C}^+ \cap C) \setminus C^+) \cup (\mathbf{C}^- \cap C). \end{aligned}$$

Finally, replacing C^- in Eq. 2 with the final rewrite obtained for C^- in Eq. 3 yields $C^- \setminus \mathbf{C}^- = (\mathbf{C}^+ \cap C) \setminus C^+$. □

THEOREM 3. *The set of false negatives* $\mathbf{C}^- \setminus C^-$ *equals* $\mathbf{C}^- \setminus C$.

PROOF. Because $C^+ \cap \mathbf{C}^- = \emptyset$, we can obtain the rewrite

$$\begin{aligned} \mathbf{C}^- \setminus C^- &= \mathbf{C}^- \setminus (C^- \cup C^+)\\ &= \mathbf{C}^- \setminus C \end{aligned}$$

□

These two theorems are useful for CDE learning because they reveal the relationships between false positives and the set of word pairs co-occurring in positive examples C^+ and between false negatives and the set of word pairs C. In particular, they imply that false positives and false negatives can be reduced by *enlarging* C^+ and C, respectively.

Intuitively, Theorem 2 captures that when using the set C to derive CDEs, the set of incorrect CDEs comprises elements in the intersection of $\mathbf{C}^+$ and C. That is, elements in C are considered as CDEs even though they actually co-occur in $\mathbf{C}^+$. However, this set is further reduced using elements in C^+. Namely, those CDEs found to co-occur in C^+ are discarded because they are incorrect. In other words, C^+ is used to filter incorrect CDEs. The larger this set, the larger is the number of incorrect CDEs (false positives) that can be filtered. Likewise, Theorem 3 can be interpreted as follows: CDEs are derived from C. Elements, whether correct or not, cannot be recognized as CDEs when they are not in C. Thus, enlarging C reduces the number of correct CDEs that cannot be derived (false negatives).

To achieve the learning goals, we now discusss the use of self-matches and self-training to enlarge C^+ and C, respectively.

3.3 Using Self-Matches as Examples

We enlarge C^+ by taking into account the following observation: every pair of instances formed by one instance and itself, i.e. instance pairs of the form (n, n) called *self-matches*, is correct.

Based on this observation, the given set of positive examples M^+, is augmented with self-matches, denoted as $M'^+ = \{(n_i, n_i) | n_i \in N\}$. Word pairs co-occurring in M'^+, referred to as C'^+, help to complement the word pairs C^+ extracted from M^+ because M'^+ and M^+ represent complementary sources. Intuitively, attribute values of the same instance contain *related words* whereas two different instances referring to the same thing, might used *similar words* (e.g. synonyms). That is, C^+ is a source of word pairs representing similar words that might be not available in C'^+, and C'^+ contains related words not captured by C^+.

EXAMPLE 3. *Figure 1(a) and Fig. 1(b) show two WCGs that are constructed from the positive examples* $M^+ = \{(n_1, n_2), (n_6, n_7)\}$ *and the self-matches* $M'^+\{(n_1, n_1), \cdots, (n_8, n_8)\}$*, respectively. They capture different sets of word pairs co-occurring in* M^+ *and* M'^+ *(solid lines) and different false positives (dotted lines). For example, note the pair of related words* (`apple`, `intel`) *and the pair of similar words* (`laptop`, `notebook`) *only co-occur in* M'^+ *and* M^+*, respectively. These word pairs will be incorrectly derived as CDEs (false positives) if only* M^+ *or* M'^+ *is used.*

Instead of using $C^- = C \setminus C^+$ (Eq. 1), we thus compute CDEs by considering the union of C^+ and C'^+ as

$$C^- = C \setminus (C^+ \cup C'^+), \quad (4)$$

where as defined before, C is the Cartesian product representing all possible pairs of words in W_{M^+}, and C^+ and C'^+ are all word pairs co-occurring in M^+ and M'^+, respectively.

In this way, the set of false positives is reduced from $C^- \setminus \mathbf{C}^- = (\mathbf{C}^+ \cap C) \setminus C^+$ to

$$C^- \setminus \mathbf{C}^- = (\mathbf{C}^+ \cap C) \setminus (C^+ \cup C'^+). \quad (5)$$

Note that according to Theorem 3, false negatives can be reduced if we change C in Eq. 4. Instead of using all the words contained in the values of instances in M^+, we could replace C by C' to calculate the CDEs as

$$C^- = C' \setminus (C^+ \cup C'^+), \quad (6)$$

where C' is the Cartesian product of words in $W_{M'^+}$ (words in M'^+). Note that M'^+ captures **all** instances, which include the sets of instances covered by M^+ (and $\mathbf{M}^+$). Thus, the set of all word pairs derived from M'^+, C', must be a superset of C (and $\mathbf{C}$). In other words, C' is larger than C, thus computing CDEs via Eq. 6 helps to reduce the number of false negatives.

However, Eq. 6 can also result in a larger set of false positives, which can be calculated according to Theorem 2 as:

$$C^- \setminus \mathbf{C}^- = (\mathbf{C}^+ \cap C') \setminus (C^+ \cup C'^+) = \mathbf{C}^+ \setminus (C^+ \cup C'^+) \quad (7)$$

In fact, computing the difference between Eq. 5 and Eq. 7, we obtain $(\mathbf{C}^+ \setminus C) \setminus C'^+$, which represents the additional set of false positives produced by Eq. 6.

Intuitively, Eq. 1 uses all possible pairs of words contained in the values of positive examples as candidate CDEs (C). Then, word pairs that actually co-occur in positive examples (C^+) are used to discard incorrect CDEs. Eq. 4 goes one step further to also discard those related word pairs that co-occur in the same instance (C'^+). In Eq. 6, we also enlarge the set of candidates (C' instead of C). This set is however too large, including all word pairs that co-occur in positive examples C (in all matches $\mathbf{C}$) as well as those that do not. Hence, a larger number of correct CDEs can be learned, but the result also includes more incorrect CDEs. Since the effect of using Eq. 6 is not unambiguously positive, we only use self-matches to reduce false negatives, i.e. Eq. 4.

3.4 Enriching Examples with Self-learning

The previous discussion suggests that in order to reduce false positives as well as false negatives, candidate word pairs cannot be chosen randomly but shall co-occur in matches. Self-learning is employed to enrich the given set of positive

examples with more matches, so that C is "selectively" enlarged to reduce the number of false negatives.

For this purpose, we assume a black-box matcher for candidate selection. It could be based on boolean matching, which given the instances N, produces the candidate set M_i at iteration i. The only property we assume is that this matcher can be parametrized to produce different sets of matches $\{M_1, \ldots, M_m\}$ that vary in size. In particular, we assume a list in which these sets are sorted:

DEFINITION 10 (SORTED CANDIDATE SETS). *Let C_i be the Cartesian product of words in M_i. There is a list of candidate sets $\{M_1, \ldots, M_m\}$, where $M_i \subseteq M_{i+1}$ such that $C_i \subseteq C_{i+1}$, $\forall 1 \leq i < m$.*

With this list, we can guarantee that by using a larger set of candidates (M_{i+1} instead of M_i), more words can be taken into account ($C_{i+1} \supseteq C_i$). Clearly, the computation of the sorted candidate sets can be naturally supported by a thresholded similarity function. It is typically monotonic w.r.t. the threshold such that a smaller threshold always yields a larger set of candidates. In our boolean matching approach, we use a boolean similarity function that is based on the word overlaps between values, $same(n[a], n'[a]) = 1$ iff $\exists w.w \in W_{n[a]} \wedge w \in W_{n'[a]}$. Since the number of word overlaps is monotonic w.r.t. the size of the resulting candidate sets, we vary this measure to obtain the sorted list.

Self-learning is an iterative process, where the following components are computed in every iteration i:

$$C_i^- = C_i \setminus (C_i^+ \cup C'^+) \quad (8)$$

$$M_{i+1}^+ = Filtering(M_{i+1}, C_i^-) \quad (9)$$

where $0 \leq i < m$, C_i is the Cartesian product of words contained in values of instances in M_i^+, C_i^+ is the set of word pairs co-occurring in M_i^+, and C'^+ is the set of word pairs co-occurring in self-matches. The result of every iteration i comprises the set of CDEs, C_i^-, the candidate set M_{i+1} and the positive examples M_{i+1}^+. The positive example M_{i+1}^+ captures the set of results obtained by filtering the candidate set M_{i+1} using the CDEs obtained from the previous iteration, C_i^-. That is, we employ two boolean functions. Whereas similarity evidences are used for candidate selection in the first step to obtain M_{i+1}, CDEs are used as dissimilarity evidences to perform a subsequent candidate filtering step to compute M_{i+1}^+. In particular, $Filtering(M_{i+1}, C_i^-)$ is based on our CDE-based dissimilarity function: for a given attribute a, an instance pair $(n[a], n'[a]) \in M_{i+1}$ is considered a non-match when there exists a CDE $(w_i, w_j) \in C_i^-$ that co-occurs in $(n[a], n'[a])$.

In the beginning, the initial set of CDEs, C_0^-, is derived directly from the provided positive examples M_0^+ and self-matches M'^+ using Eq. 4. Then, we perform candidate selection to produce the candidate set M_1. Filtering this set using C_0^-, we obtain the refined set of examples M_1^+. These steps of CDE learning, candidate selection and candidate filtering are iteratively performed until we reach the last iteration m, which yields the final set of instance pairs M_m^+.

We provide the following theorem to show that this self-learning reduces the number of false negatives, i.e. the number of correct CDEs learned in iteration $i+1$ is a superset of the correct CDEs learned in iteration i:

THEOREM 4. *For any two sets of CDEs C_i^- and C_{i+1}^- learned from M_i^+ and M_{i+1}^+, respectively, we have*

$$M_i^+ = M_{i+1}^+ \cap M_i \quad (10)$$

$$C_{i+1}^- = C_i^- \cup (C_{i+1}^- \setminus C_i). \quad (11)$$

PROOF. Firstly, note all instance pairs that are filtered from M_i must contain at least one CDE in C_i^- and all instance pairs in M_i^+ must contain no CDEs in C_i^- such that the result of filtering on M_i using CDEs in C_i^- is M_i^+. Given M_i is a subset of M_{i+1} ($M_i \subseteq M_{i+1}$), we can write $M_{i+1} = M_i \cup (M_{i+1} \setminus M_i)$. Thus, the result of filtering on M_{i+1} using C_i^- can be calculated as $M_{i+1}^+ = M_i^+ \cup (M_{i+1} \setminus M_i)^+$, where $(M_{i+1} \setminus M_i)^+$ captures the result of filtering on $(M_{i+1} \setminus M_i)$. As a result, we have $M_i^+ = M_{i+1}^+ \cap M_i$ and $C_i^+ \subseteq C_{i+1}^+$ such that

$$C_{i+1}^+ \cap C_i^+ = C_i^+. \quad (12)$$

Secondly, since no CDEs in C_i^- can co-occur in M_{i+1}^+, the intersection of C_i^- and C_{i+1}^+ is empty:

$$C_{i+1}^+ \cap C_i^- = \emptyset \quad (13)$$

Then, we can rewrite C_{i+1}^- as the union of two terms:

$$C_{i+1}^- = (C_{i+1}^- \cap C_i) \cup (C_{i+1}^- \setminus C_i) \quad (14)$$

where $C_{i+1}^- \cap C_i$ are CDEs in C_i and $C_{i+1}^- \setminus C_i$ are CDEs not in C_i. Next, $C_{i+1}^- \cap C_i$ can be calculated as the complement set of word pairs that co-occur in C_{i+1}^+ and C'^+ as:

$$C_{i+1}^- \cap C_i = C_i \setminus ((C_{i+1}^+ \cap C_i) \cup C'^+) \quad (15)$$

Note $C_i = C_i^+ \cup C_i^-$, then Eq. 15 can be further rewritten as

$$C_{i+1}^- \cap C_i = C_i \setminus ((C_{i+1}^+ \cap C_i^+) \cup (C_{i+1}^+ \cap C_i^-) \cup C'^+). \quad (16)$$

Substituting Eq. 12 and Eq. 13 into Eq. 16, we can rewrite Eq. 16 according to Eq. 8 as follows:

$$\begin{aligned} C_{i+1}^- \cap C_i &= C_i \setminus ((C_{i+1}^+ \cap C_i^+) \cup (C_{i+1}^+ \cap C_i^-) \cup C'^+) \\ &= C_i \setminus (C_i^+ \cup C'^+) \\ &= C_i^- \end{aligned} \quad (17)$$

Finally, substituting Eq. 17 into Eq. 14, we prove that $C_{i+1}^- = C_i^- \cup (C_{i+1}^- \setminus C_i)$ □

Intuitively, Theorem 4 captures that the CDEs and filtering result that are derived from a larger candidate set M_{i+1} are composed of two parts: *all* the CDEs and filtering result that could already learned with M_i and some new CDEs and result that are possible with the new words and instance pairs in M_{i+1}.

For a better understanding of the effects of using self-matches and self-learning, we refer to our set-theoretic analysis in the the Appendix.

3.5 On the Combination of Thresholded and Boolean Matching

We completely rely on boolean functions to obtain a simple, threshold-free approach that does not require fine-tuning. However, there are two natural ways to combine this boolean approach with thresholded instance matching.

A threshold similarity function can be used to generate the sorted list of candidate sets as previously discussed. Candidate sets are computed for and sorted according to the

given threshold. In this case, thresholded matching is used for the candidate selection step, which is then followed by a boolean candidate filtering step based on CDEs. As a whole, this combination represents a matching solution that implements our iterative embedded process of CDE learning, thresholded candidate selection and boolean candidate filtering.

This whole solution can also be treated as a black box for *boolean filtering*. Given candidates computed by a thresholded approach, it can be used to identify and filter dissimilar matching candidates.

3.6 Multiple Attributes

So far, we have discussed CDE learning and instance matching always for a given attribute a. That is, when some word pairs are said to co-occur in values of some instances, it actually means they co-occur in the values of an specific attribute a of some instances. Because CDEs learned from different attributes capture dissimilarity evidences from different aspects, a non-matching instance pair that cannot be identified by CDEs learned from one attribute could be identified by CDEs learned from other attributes. In the multiple attributes setting, we apply the self-learning procedure to every attribute. That is, CDEs are learned for any attribute. A candidate match is considered correct only if it cannot be filtered by CDEs that are learned from every attribute. Let $\{a_1, a_2, \cdots, a_k\}$ be a set attributes and M_{ij}^+ be the result obtained by applying the filtering step on the candidate set M_i using CDEs learned from a_j, we calculate the result as the intersection of the results obtained for every attribute, i.e. $M_i^+ = \bigcap_{j=0}^{k} M_{ij}^+$.

4. IMPLEMENTATION

We previously focus on the main ideas behind CDE learning. Here, we also consider the aspect of efficiency, presenting a solution that aim to address the following two problems:

Duplicate Instance Pairs: Self-learning is an iterative procedure in which the steps of CDE learning, candidate selection and filtering. Clearly, candidate sets processed in different iterations largely overlap in the instance pairs they contain. It is thus not efficient to repeatedly process the same instance pairs over several iterations. We can leverage Theorem 4 to solve this problem: for any two adjacent sorted candidates sets M_i and M_{i+1}, we know that the CDEs and filtering result computed for the same set of instance pairs in M_i and $M_{i+1} \cap M_i$ are the same, i.e. $M_i^+ = M_{i+1}^+ \cap M_i$, $C_i^- = C_{i+1}^- \cap C_i$. Since the computed results for instance pairs in M_i in each iteration are the same, we can simply remove the instance pairs in M_i from M_{i+1}, i.e. only process the instance pairs in the set $M_{i+1} \setminus M_i$.

Expensive WCG Calculation: To learn CDEs according to Eq. 8, the Cartesian product of all words in the candidate set, C, is taken into account. Then, we consider C^+ to derive the CDEs as the complement set C^-. While using these different sets help to explain the idea, we observe that for CDE-based filtering, only C^+ is actually needed. According to Theorem 1, a word pair is a CDE if it is not in C^+. Thus, an instance pair can be considered a non-match if one of its word pairs is not in C^+ (i.e. that word is a CDE). In this way, we can check for CDEs and perform filtering simply by looking at word pairs that co-occur in a given instance pair and word pairs in C^+. To capture this idea, we can redefine the CDE-based dissimilarity function as follows:

DEFINITION 11 (C-BASED DISS. FUNCTION). *Let M^+ be a set of positive examples and C^+ be the set of word pairs that co-occur in M^+, the* CDE-based dissimilarity function $\neg same : N[a] \times N[a] \rightarrow \{0, 1\}$ *maps a pair of attribute value $(n[a], n'[a])$ to 1, if there exists a word pair (w_i, w_j) that co-occurs in $(n[a], n'[a])$ and $(w_i, w_j) \notin C^+$, and 0 otherwise.*

Since the number of matches is usually much smaller than the number of non-matches, we can expect that most word pairs do not co-occur in matches, i.e. the size of C^+ is much smaller than C^- and C. Thus, focusing on C^+ could largely improve efficiency.

The procedure for instance matching, which assumes a sorted list of candidate sets obtained via candidate selection and iteratively performs CDE learning and candidate filtering, is shown in Alg. 1. Due to the change discussed above, CDE learning and candidate filtering now involves the use of C^+ instead of C^-. That is, Alg. 3 actually does not learn CDE but C^+, which is then used to identify CDE in instance pairs and to filter them.

First, C^+ is constructed using self-matches and the given training examples (line 1-3). Then filtering (Alg. 2) and CDE learning (Alg. 3) are applied iteratively on each of the sorted candidates sets (line 5-11). As discussed, in Alg. 1, in each iteration, the instance pairs in M_{i-1} can be removed from M_i first to avoid duplicate processing. The resulting set of candidates is used to initiate M_i^+ (line 6). Then, non-matches from this candidate set are filtered according to Def. 11 (line 8). After filtering using all attributes, enrichment is performed to collect more words and C_i^+ from the filtering result M_i^+ (line 10). Finally, the filtering result for every iteration is added to the result set (line 11).

Algorithm 1: Self-learning based instance matching

Input: M_0^+, the initial examples; M'^+, self-matches; $\{M_1, M_2, \cdots, M_m\}$, sorted candidates sets; and the attributes $\{a_i, a_2, \cdots, a_k\}$.
Data: C_{i,a_j}^+, word pairs co-occurring in the values of a_j of instances in M_i^+
Result: M^+.

1 //■Learn CDEs■ with self-matches + examples;
2 **for** $j = 1$ **to** k **do**
3 $\quad C_{0,a_j}^+ := LearnCDE(a_j, M_0^+ \cup M'^+)$;
4 //Self-training;
5 **for** $i = 1$ **to** m **do**
6 $\quad M_i^+ = M_i \setminus M_{i-1}$;
7 $\quad$ **for** $j = 1$ **to** k **do**
8 $\quad\quad M_i^+ := Filtering(a_j, M_i^+, C_{i-1,a_j}^+)$;
9 $\quad$ **for** $j = 1$ **to** k **do**
10 $\quad\quad C_{i,a_j}^+ := C_{i-1,a_j}^+ \cup LearnCDE(a_j, M_i^+)$;
11 $\quad M^+ := M^+ \cup M_i^+$;
12 **return** M^+;

EXAMPLE 4. *Fig. 2 illustrates the use of* `Title` *for the data in Tab. 1. Lets have a look at iteration 1: we have already collected the word pairs C_0^+ and C'^+ that co-occur in the examples M_0^+ and the self-matches M'^+ respectively.*

Algorithm 2: Filtering

Input: Attribute a_j; and candidates set M_i, word pairs C^+_{i-1,a_j}.

Result: A set of instance pairs M_i^+.

```
1 foreach (n, n') ∈ M_i do
2     foreach w ∈ W_{n[a_j]} do
3         foreach w' ∈ W_{n'[a_j]} do
4             //if (w, w') is a CDE;
5             if (w, w') ∉ C^+_{i-1,a_j} then
6                 M_i := M_i \ {(n, n')} ;
7 return M_i^+ := M_i;
```

Algorithm 3: Learning CDEs

Input: attribute a_j; instance pairs M_i^+.

Result: C^+_{i,a_j}, word pairs co-occurring in the values of a_j of instances in M_i^+.

```
1 //Find word pairs co-occurring in M_i^+;
2 foreach (n, n') ∈ M_i^+ do
3     foreach w ∈ W_{n[a_j]} do
4         foreach w' ∈ W_{n'[a_j]} do
5             C^+_{i,a_j} := C^+_{i,a_j} ∪ {(w, w')} ;
6 return C^+_{i,a_j};
```

Now the algorithm applies filtering to the larger candidate set M_1, and then learns CDE from M_1^+, as follows:

*1. **Removing duplicate instance pairs**: The pairs in M_0 are removed from M_1 to avoid duplicate processing.*

*2. **Filtering**: Filtering is applied to the remaining instance pairs in M_1. For example, (n_1, n_3) is derived as a non-match according to Def. 11 because the word pair $(\texttt{i5}, \texttt{i7})$ co-occurs in (n_1, n_3) but not in C_0^+ and C'^+. Finally, we can infer (n_3, n_4) as a match, which is then used as a positive example for CDE learning in the next step.*

*3. **Learning CDEs**: Enrichment is performed with the pairs of new words taken from M_1^+ such as $(\texttt{apple}, \texttt{retina})$ and $(\texttt{ME664LL}, \texttt{pro})$ that co-occur in (n_3, n_4). As a result, C_1^+ contains all the word pairs that co-occur in M_0^+, M'^+ and M_1^+.*

The time complexity for every iteration in Alg. 1 depends on the number of word pairs that are compared. The algorithm needs to check all the instance pairs in M_0^+, M'^+ and M_m, where M_0^+ denotes the initial set of positive examples, M'^+ is the set of self-matches, and M_m denotes the largest one among all the sorted candidate sets (note that M_m contains all the examples added to the initial set as part of the iterative enrichment). For each instance pair, let k be the number of attributes and w^2 be the maximum number of word pairs that co-occur in the value of an attribute. In worst case, the number of word pairs that the algorithm processes is $(|M_0^+ \cup M'^+ \cup M_m|)kw^2$. With m as the number of iterations, the total time complexity is $O((|M_0^+ \cup M'^+ \cup M_m|)kw^2m)$.

5. EXPERIMENTAL EVALUATION

To study the proposed solution, we employ a recent instance matching benchmark [9] that captures data from enterprise databases as well as synthetic data. Tab. 3 provides an overview of the datasets.

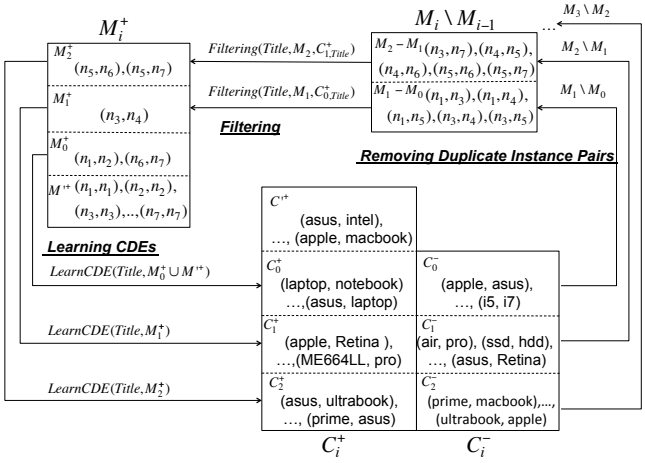


Figure 2: An example of the algorithm

Table 3: Instances and ground truth (GT).

Task	Instances		GT
	Dataset1	Dataset2	
AD	2,616	2,294	2,224
AB	1,081	1,092	1,097
Rest	864		112

ACM-DBLP (AD). These datasets in the benchmark [9] include well-structured bibliographic data from DBLP and ACM digital library. This one is manually created and thus, is of higher quality among all datasets. Therefore, the task represented by this is of low difficulty.

Abt-Buy (AB). This matching task in the benchmark [9] is performed between instances collected from `http://abt.com` and `http://buy.com`. There are many errors and noises that cannot be avoided during automatic data collection. This matching task is in fact the most difficult one in this experiment.

Restaurant(Rest). This dataset is made available through the OAEI 2010[1] benchmark. We removed the `telephone` attribute because with it, previous results published by the benchmark show that it is too easy of an matching task (all systems perform well).

Systems. We compare our techniques with three recent approaches [14, 13, 8] proposed for instance matching. Two of them are thresholded instance matching approaches, denoted as `SiFi` [14] and `ST` [8]. `SiFi` learns similarity functions and thresholds from positive and negative examples. `ST` learns the most discriminative attributes and values to be used for matching. It uses a given similarity function `ISub` and thresholds manually set by experts. The third, named `Paris`, is a boolean instance matching approach that outputs true for two instances, if they have the same attribute values. The idea here is to iteratively cross-fertilize instance matching results with schema matching results until convergence is reached [13].

Quality Metrics. We use the standard metrics for comparing instance matching results: Precision (P) and Recall

[1] `http://oaei.ontologymatching.org/2010`

(R) and F-measure (F).

Setting. All experiments were run on a server with two Intel Xeon 2.8GHz Dual-Core CPUs, using 8GB of main memory, running Linux with kernel version 2.6.18. The presented results are computed as an average over five runs.

5.1 Parameter Analysis

Compared to thresholded instance matching approaches, our solution is parameter-free in this sense: we simply use all the attributes, value overlap as the boolean function for candidate selection, and $m = 10$ as the number of iterations. As illustrated in Fig 3(d) (average F-measure for all tasks), our approach achieves fairly stable results for different values of m. The only exception is when m is set to 1, which means the CDEs are learned only based on the provided examples and self-matches. Self-learning improves the results (changing $m = 1$ to any value great than 1). However, the results suggest improvements converge quickly after $m > 4$.

Now, we study the sensitivity of existing approaches (`SiFi` and `ST`) with respect to training data and parameters. We will focus our discussion on the task `Rest`.

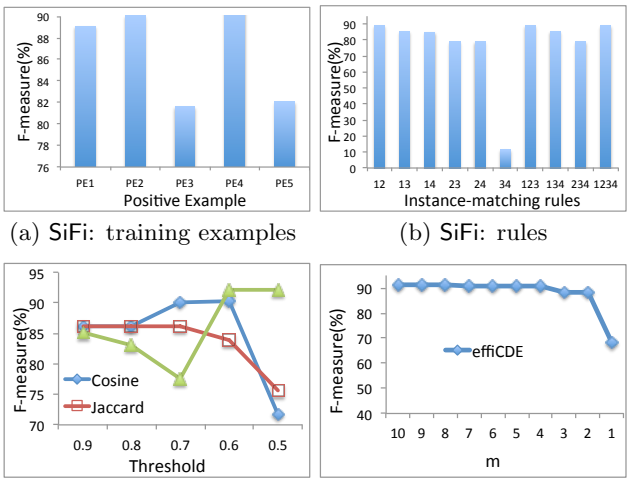
SiFi. In the experiments, we adopt the four instance-matching rules as reported by the authors in their original paper [14]. SIFI requires attributes to be manually set for each rule. Also, it requires the similarity functions and thresholds to be manually set for some attributes so that they can be learned for other attributes. Besides this problem of manual tuning, the sensitivity of the approach w.r.t. parameters and training data is as follows.

Firstly, different sets of positive examples of the same size (30% of the ground truth) were randomly selected to analyze the sensitivity of training data. As illustrated in Fig 3(a), we can see results achieved by `SiFi` were not stable. Depending on the set of training examples, results vary between 81% and 9% in term of F-measure.

Secondly, different combinations of the four original instance-matching rules were used. For example, in Fig. 3(b), we use `13` to denote the usage of the first and the third rules. We can see that also with respect to parameters, the results are not stable (vary between 78% and 90%, not counting outlier 34). Considering 34, F-measure result might be as low as 10%. We note the results are the same for the settings `12` and `1234`, indicating that the third and the fourth rules are actually not necessary for the task at hand. However, it is difficult for experts to determine and select the rules (especially when not only result quality but also efficiency is a relevant factor).

Finally, we analyze the effect of using different thresholds and similarity functions that we manually set for the attribute `name`. As illustrated in Fig. 3(c), results greatly vary when either the similarity function or the threshold changes. Note in practice, experts have to chose the best setting for multiple attributes, selecting from more than tens of relevant similarity functions and all candidate thresholds in the $[0, 1]$ interval.

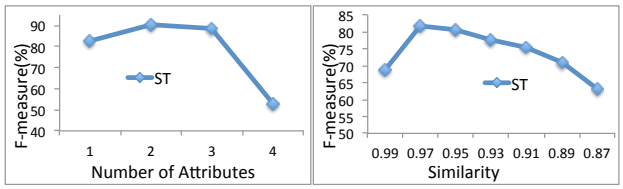
ST. With this approach, experts have to select the attributes used for instance matching so that instance pairs, whose similarity on any attribute is higher than the threshold can be considered as matches. Intuitively, more matching instance pairs can be found when we consider more attributes (high recall). However, a high number of attributes may also result in too many non-matches (low precision). In Fig. 4(a), we see this for the data in our experiment.



(a) SiFi: training examples (b) SiFi: rules

(c) SiFi: similarity functions and thresholds (d) effiCDE: m

Figure 3: Parameter analysis for SiFi and effiCDE.



(a) Different no. of attributes (b) Different thresholds

Figure 4: Parameter analysis for ST

F-measure result initially improves as the number of attributes increases (the effect of improved recall) and then drops sharply as many more attributes are added (the effect of reduced precision).

We also analyze the results computed using different thresholds that are between 0.87 and 0.99. As shown in Fig. 4(b), `ST` is extremely sensitive to the chosen threshold. For example, F-measure decreases sharply from 81% to 68% when the threshold is slightly changed from 0.97 to 0.99.

In summary, these experiments show that while good results can be achieved using thresholded approaches (approaches that require fine tuning, often much more than just the thresholds), they are very sensitive to the choice of parameters and data. Result quality depends not only on the underlying approach but also the effectiveness of the experts applying it. In what follows, we now compare these thresholded approaches with boolean approaches, including our approach and `Paris`, which aim to reduce the efforts needed for manual tuning.

5.2 Efficiency of Instance Matching

In this section, we compare the efficiency of our solution with existing approaches. As discussed in Sec.3.5, it can be combined with the use of thresholds in two different ways. When not explicitly mentioned, experimental results presented for our solution here are entirely based on boolean matching as presented in the paper.

Because it is sometime difficult to separate the training and matching processes in instance matching approaches (they are often intertwined as results from matching are used

to improve learning), we evaluate the efficiency of all methods w.r.t. total running time as shown in Tab. 4.

CDE and effiCDE represent our solution with and without efficiency improvements as proposed in Sec. 4. Clearly, effiCDE achieved much better performance: it needs only 2% of the time taken by CDE. Especially, because the size of C^+ (used by effiCDE) is only 1% - 2% of the size of C^-(used by CDE) in the experiments, effiCDE is also much more efficient in terms of memory usage.

Comparing to the thresholded instance matching approaches SiFi (ST), effiCDE exhibits better (similar) overall performance. In the learning step, effiCDE is efficient because it avoids the learning of complex instance matching rules: it only has to focus on word pairs that co-occur in a relatively small-amount of matches. It is also efficient in the matching step because instead of possibly complex similarity calculations (depending on the similarity functions), it only requires boolean matching on words. For the dataset AD as an example, effiCDE requires only 43 seconds, which is only 10% of the time required by SiFi. SiFi exhibits worst performance because it has to search the best rules from a large search space of rule candidates. Even though SiFi implements a strategy for eliminating candidates with redundant similarity functions and thresholds, we observe that the operations needed for determining and removing redundancy itself is still too costly. ST is fast because compared to SiFi, it pursues a more easy learning task, which does not require the learning of similarity functions and thresholds (hence, much smaller search space).

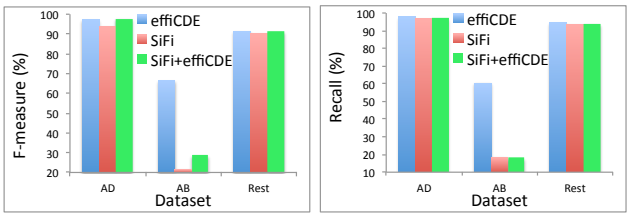
Paris is the fastest among all approaches. This is because just like our approach, PARIS only employes boolean matching. It is faster than our solution mainly because it is much more aggressive in pruning candidates. In our approach, candidates are recognized whenever they match on some words contained in their values. Further, before filtering, some are actually "re-examined" in several rounds of our iterative process. Paris keeps only candidates that match on their values and filters them more aggressively throughout the iterations. While this strategy largely increases performance, it also leads to lower recall when the data contains more noises, as discussed in the following.

Table 4: Performance of learning and instance matching in ms.

	effiCDE	CDE	SiFi	ST	PARIS
AD	42,552	2,139,949	502,450	48,362	**25,955**
AB	14,068	699,031	33,418	12,202	**7,180**
Rest	**4,508**	52,798	58,491	5,471	10,517

5.3 Effectiveness of Instance Matching

Tab. 5 presents results for our three quality metrics. Overall, we can see that effiCDE outperforms all the other approaches in terms of F-measure. Compared to the second best result achieved by any of the studied approaches, it improves F-measure by +3.66%, +206.47% and +1.34% for AD, AB and Rest, respectively. Especially, we note effiCDE is most effective in filtering non-matching instance pairs. This is reflected in the higher precision results: the improvements over the second best are +6.51%, 181.64% and 1.56% for AD, AB and Rest, respectively. We note that in fact, our filtering approach primarily aims at precision. Regarding recall, the boolean matching as implemented for the candidate selection step in our approach is too strict as it only select



(a) F-measure (b) Recall

Figure 5: effiCDE as Filters

exact matches. This however, can be remedied by using thresholded matching for candidate selection as discussed in Sec. 3.5 and in what follows.

Regarding specific tasks, we observe effiCDE is particularly effective in dealing with the most difficult task, i.e. finding product pair matches on the relatively noisy AB dataset. The results achieved for this task are much worse then for other tasks. This is because the automatically extracted product descriptions largely vary, while the bibliographic data AD and restaurant data Rest are more structured are uniform. As a result, when entire attribute values are considered, matches are hard to find on AB. In fact, at this attribute-level, AB cannot provide sufficient similarity evidences such that the thresholded instance matching approaches SiFi and ST fail to find the thresholds that cover the given example matches. At this level, there are also not sufficient evidences for a boolean approach: PARIS incorrectly prunes candidates because only few product pairs entirely match on their attribute values. More evidences can however be collected at the word level. Products that have low similarities on entire attribute values, often share very distinctive word pairs. Our CDE-based solution leverage this, providing a better solution for noisy data by exploiting more fine-grained dissimilarity evidences. The F-measure improvement for AB is +206.34%.

Combining Boolean and Thresholded Matching. Instead of using a pure boolean approach, we also experiment with the combination of effiCDE and thresholded matching. It is applied for filtering results computed by other approaches. Fig. 5(a) shows results obtained by applying effiCDE as a filter on SiFi results. Compared to SiFi, F-measures consistently increase for all tasks after applying effiCDE as a filter. We can see in Fig. 5(b) that recall does not change. Thus, this F-measure improvement is entirely due to the positive effect of filtering on precision. As a filter, effiCDE helps to reduce non-matches while preserving all matches produced by the underlying approach.

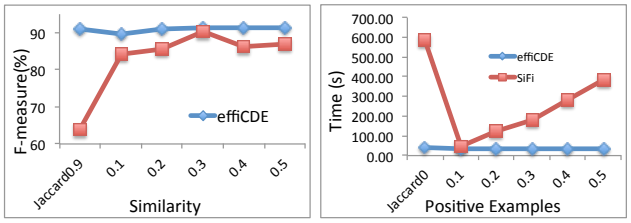
However, this combination is not always better than effiCDE. SiFi could not find many candidates for the noisy AB datasets. Using effiCDE as a filter could not solve this problem of recall.

5.4 Labeling Effort

Because SiFi is the only approach that requires training examples (other approaches used "pseudo-examples", i.e. matches computed in the first run), we use it as a comparison to study the labeling efforts needed to obtain examples. As example, we use different sets of matches (between 10% and 50%) taken from the ground truth. To consider sensitivity w.r.t. noises in the training data, we also use the set of instance pairs whose *Jaccard* similarity is greater than 0.9 as pseudo-examples, denoted as Jaccard0.9. For

Table 5: Effectiveness of instance matching in terms of P, R and F

	effiCDE			SiFi			ST			PARIS			chg% over second best		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
AD	**96.58**	**97.84**	**97.21**	90.68	97.10	93.78	85.19	96.99	90.71	93.20	93.62	93.41	+6.51%	+0.76%	+3.66%
AB	**74.24**	**60.43**	**66.63**	26.36	18.51	21.75	20.77	18.69	19.67	12.01	2.55	4.21	+181.64%	+226.47%	+206.34%
Rest	**88.33**	**94.64**	**91.37**	86.77	93.75	90.13	87.83	90.17	88.98	5.90	45.53	10.45	+1.56%	+0.94%	+1.34%



(a) Effectiveness (b) Efficiency

Figure 6: Evaluation result for different labeling efforts.

`Jaccard0.9`, precision, recall, and F-measure are 89.89%, 79.46% and 84.36%, respectively.

Figure. 6(a) shows the F-measures results. Clearly, `effiCDE` manages to maintain high quality results for varying amount of labeling efforts, including for `Jaccard0.9` that requires no efforts. This is because training examples are largely enriched by self-matches as well as matching obtained via self-training. Thus, the initial set of seeds is not critical for our approach. Note that `effiCDE` performs well for `Jaccard0.9`, while `SiFi` provides very low quality results. The non-matches in these pseudo-examples represent false similarity evidences, which have a large negative impact on the thresholds learned by `SiFi`. With `effiCDE`, word pairs that should be CDEs are incorrectly recognized as word pairs co-occurring in matches, due to the existence of non-matches in the examples. This only results in a smaller set of initial CDEs that can be learned. Even with this lack of CDEs, a non-match can still be filtered for two reasons: 1) there are multiple CDEs existing in various attributes of a non-match and any attribute can be used for filtering; 2) most importantly, many more CDEs can be derived later through the enrichment via self-matches and self-training.

As shown in Fig. 6(b), `effiCDE` achieves stable running time given varying amount of labeling efforts while `SiFi` takes more time when the number of training examples increases. As discussed, the time complexity of `effiCDE` depends on the number of instance pairs in the initial set of examples, self-matches and sorted candidates sets. The running time of our solution is less sensitive to the initial training examples because its size is relatively small compared to the other two factors. `SiFi` however directly depends on it. An increase in size leads to a much larger search space that has to be considered for rule learning.

6. RELATED WORK

Typically, instance matching is solved in two steps where firstly, a relatively simple type of approaches is applied first to quickly select candidates. This high recall but low precision type is used for the candidate selection (blocking) step, which precedes a more sophisticated filtering step involving complex instance matching rules.

Candidate selection approaches partition instances into blocks containing similar instances [3, 16, 11]. Examples, such as PPJoin+ [17] and suffix arrays based blocking [6], typically use boolean functions, grouping instances in the same block if their entire values (or subparts such as prefixes) are the same. Our approach is similar to blocking in that it also uses boolean evidences. However, they are more fine-grained word-level evidences that help to produce blocks much smaller then what blocking approaches compute. In fact, we apply blocking based on value overlap to obtain candidates, and use our approach to filer them to achieve higher precision.

For fine-grained matching and filtering based on thresholds, different similarity functions have been incorporated into matching rules, including character-based metrics (e.g. edit distance) and token-based metrics that consider the rearrangements of words [7]. `Paris` [13] measures the degrees of matching based on probability estimates (and cross-fertilizes them through evidence propagation between the data and schema level).

For learning the rules, there are supervised techniques based on existing machine learning methods (such as SVM [4]) or specific ones designed for instance matching (e.g. learning similarity function predicates from training data can be reduced to the maximum rectangle problem [5]). As shown for `SiFi` [14], highest quality can be achieved when combinations of attributes, similarity functions and thresholds are considered for rule learning. Further, self-learning, which incorporates previous learning results into the loop can yield improvements. Recent approaches for this include `Paris` that employes evidence propagation [13] and `ST` [8].

There are also semi-supervised techniques based on probabilistic graphical models. It has been shown that many existing instance matching (and blocking) approaches can be captured as Markov Logic formulas and reformulated as a Markov Logic learning problem [12]. However, this involves learning both the structure (formulas) and their weights. Existing works focus on weight learning such that the formulas have to be specified manually.

Our idea of using dissimilarity evidences for filtering is most related to the use of constraints also called negative rules. It has been shown that using manually designed constraints can help to effectively filter non-matches [15, 1]. The main differences to our work are: (1) these constraints (just like instance matching rules) capture evidences at the level of attributes, i.e. they are based on comparing the whole attribute values. (2) Constraints (which are relative small in number) shall capture the domain knowledge of experts while our dissimilarity functions (a large number of them) are driven by the data. They are manually designed by experts while our functions are learned from the data.

7. CONCLUSION

For the problem of instance matching, we provide a solution that instead of threshold functions that capture attribute-level evidences, employes more simpler boolean functions but also more fine-grained word-level evidences.

We show that with this boolean approach, a parameter-free procedure can be designed that when combined with word-level evidences, can be highly effective. Compared to state-of-the-art instance matching approaches, this solution greatly improves result quality and most importantly, is not sensitive to the choice of training data and parameter. We show that it can be employed as a standalone solution for instance matching or combined with a matcher as a boolean filter. As future work, we will explore the usage of more complex boolean functions as well as a tighter combination of the boolean and threshold approaches.

8. REFERENCES

[1] A. Arasu, C. Ré, and D. Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE*, pages 952–963, 2009.

[2] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.

[3] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, pages 87–96, 2006.

[4] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.

[5] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. Example-driven design of efficient record matching queries. In *VLDB*, pages 327–338, 2007.

[6] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, pages 305–314, 2009.

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

[8] W. Hu, J. Chen, and Y. Qu. A self-training approach for resolving object coreference on the semantic web. In *WWW*, pages 87–96, 2011.

[9] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.

[10] Y. Ma and T. Tran. Typimatch: type-specific unsupervised learning of keys and key values for heterogeneous web data integration. In *WSDM*, pages 325–334, 2013.

[11] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *WSDM*, pages 535–544, 2011.

[12] P. Singla and P. Domingos. Entity resolution with markov logic. In *ICDM*, pages 572–582, 2006.

[13] F. M. Suchanek, S. Abiteboul, and P. Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *PVLDB*, 5(3):157–168, 2011.

[14] J. Wang, G. Li, J. X. Yu, and J. Feng. Entity matching: How similar is similar. *PVLDB*, 4(10):622–633, 2011.

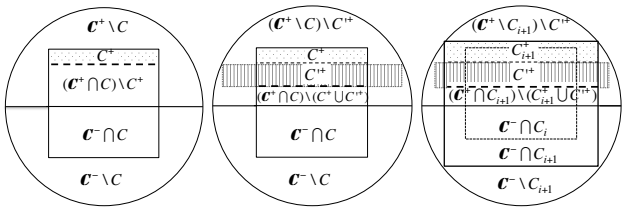
[15] S. E. Whang, O. Benjelloun, and H. Garcia-Molina. Generic entity resolution with negative rules. *VLDB J.*, 18(6):1261–1277, 2009.

[16] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD Conference*, pages 219–232, 2009.

[17] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.

APPENDIX

A. SET-BASED ANALYSIS



(a) Positive examples (b) With Self-matches (c) With Self-matches + Self-training

Figure 7: The upper (lower half) of the circle captures word pairs in matches, $\mathbf{C}^+$ (CDEs, $\mathbf{C}^-$). The whole square (circle) represents C (C). The dotted (lined) area contains word pairs in the examples (self-matches). Because the learned CDEs are bounded by the square, the lower area outside the square and the upper blank area insides the square captures the false negatives and the false positives, respectively.

Here, we provide a set-based analysis and illustration of the effects of using self-matches and self-training.

Only positive examples. Fig. 7(a) shows CDE learning only using positive examples. The areas representing false positives ($C^- \setminus \mathbf{C}^- = (\mathbf{C}^+ \cap C) \setminus C^+$) and false negatives ($\mathbf{C}^- \setminus C^- = \mathbf{C}^- \setminus C$) are relatively large.

Self-Matches. Fig. 7(b) shows CDE learning with positive examples and self-matches. More word pairs can now be added using self-matches. This is reflected in Fig. 7(b) by the lined area, indicating that the upper blank area inside the square, i.e. the number of false positives, becomes smaller: $C^- \setminus \mathbf{C}^- = (\mathbf{C}^+ \cap C) \setminus (C^+ \cup C'^+)$. Because the square area is only determined by the positive examples, it does not change with the use of self-matches. Thus, using self-matches has no effect on false negatives.

Self-Matches and Self-Training. With self-training the number of positive examples increases. As a result, the square area in Fig. 7(c) is enlarged. This translates to a reduced amount of false negatives, i.e. from $\mathbf{C}^- \setminus C_i$ to $\mathbf{C}^- \setminus C_{i+1}$, $C_i \subseteq C_{i+1}$. The set of false positives is $C^- \setminus \mathbf{C}^- = (\mathbf{C}^+ \cap C_{i+1}) \setminus (C^+_{i+1} \cup C'^+)$.