Entity Matching: How Similar Is Similar

Jiannan Wang # Guoliang Li # Jeffrey Xu Yu * Jianhua Feng #

Department of Computer Science, Tsinghua University, Beijing, China

* *Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong, Hong Kong, China*

wjn08@mails.tsinghua.edu.cn; {liguoliang,fengjh}@tsinghua.edu.cn; yu@se.cuhk.edu.hk

ABSTRACT

Entity matching that finds records referring to the same entity is an important operation in data cleaning and integration. Existing studies usually use a given similarity function to quantify the similarity of records, and focus on devising index structures and algorithms for efficient entity matching. However it is a big challenge to define "how similar is similar" for real applications, since it is rather hard to automatically select appropriate similarity functions. In this paper we attempt to address this problem. As there are a large number of similarity functions, and even worse thresholds may have infinite values, it is rather expensive to find appropriate similarity functions and thresholds. Fortunately, we have an observation that different similarity functions and thresholds have redundancy, and we have an opportunity to prune inappropriate similarity functions. To this end, we propose effective optimization techniques to eliminate such redundancy, and devise efficient algorithms to find the best similarity functions. The experimental results on both real and synthetic datasets show that our method achieves high accuracy and outperforms the baseline algorithms.

1. INTRODUCTION

In the fields of data cleaning and integration, entity matching that identifies records referring to the same entity is an important operation. This problem is also known as duplicate detection, record linkage and merge/purge, and it has been extensively studied by different communities such as statistics, databases, and artificial intelligence (see [10] for a recent survey). For example, Figure 1 gives a set of records integrated from multiple sources. As there may have inconsistences in the data, we want to find the records that refer to the same person from the data.

In statistics and artificial intelligence, existing studies formulate this operation as a classification problem [12]. They enumerate all record pairs, represent each record pair as a feature vector, and classify these pairs as matching or non-matching. While these methods typically have good accuracy, they do not scale well for a large amount of data, since they need to classify n^2 feature vectors if there are n records.

In the database community, existing methods usually employ a rule-based method to find entities [7,11,13,14,18]. They define a set of record-matching rules to accommodate different representations of the same entity. Consider a record-matching rule "if two records have *similar* `name` and the same `tel`, they refer to the same entity" in Figure 1. For instance, records r_1 and r_7 have *similar* names and the same telephone numbers, they will be considered as the same entity. Then they utilize these rules to find entities.

The rule-based method has the following problems. Firstly, it is not straightforward to generate record-matching rules, and Fan et al. [11] proposed to use schema information to effectively generate record-matching rules, which may contain exact-match conditions (e.g., `tel`s are the same) and approximate-match conditions (e.g., `name`s are `similar`). Secondly although it can be efficient to support exact-match conditions, it is not straightforward to support approximate-match conditions. To address this problem, existing methods [3,7] focus on devising effective index structures and algorithms to efficiently support the approximate-match conditions. Thirdly, it is a big challenge to define "how similar is similar" for the approximate-match conditions. Traditional methods [11,14] usually suppose a similarity function (e.g., Edit Similarity and Jaccard) is given to decide whether two values are *similar*. However this method cannot be adapted to different applications and it calls for a new method to automatically select appropriate similarity functions in record-matching rules.

In this paper we address the third problem. Given a table with a set of records and a set of record-matching rules with unknown similarity functions and thresholds, we want to identify the best similarity functions and thresholds for effectively finding entities from the table. Note that it is rather hard to find the best similarity functions by only using record-matching rules [6], since there could be larger numbers of similarity functions and thresholds. To address this problem, we suppose that users can provide a set of positive examples (e.g., which records are known to be the same entity) and negative examples (e.g., which records are known to be not the same entity). We utilize these examples to identify the best similarity functions.

This problem has the following challenges. Firstly there are a large number of similarity functions, and even worse thresholds may have infinite values. It is rather expensive to find appropriate similarity functions and thresholds. Secondly, users may have different preferences (e.g., preferring

Proceedings of the VLDB Endowment, Vol. 4, No. 10

rid	name	tel	email	addr	gender
r_1	Jeffrey Yi	852	Jeff	Room 567B, Computer Science Department, CUHK, HongKong	Male
r_2	Jeff Lee	852-333333	JeffLee@abc.com	Room 123B, CS Dept., CUHK	Female
r_3	JeffreyYYu	852-222222	Jeffyx@mail.com	Room 759B, Engineering Building, CUHK	M.
r_4	Yu Xu	852	Jeffyx@mail.com	Room 759B, Engineering Building, CUHK	Male
r_5	Jeffrey Lee	852	Jeff	Computer Science Department, CUHK	F.
r_6	Jeffery Yi	852-111111	Jeffyi@abc.com	Room 567B, Computer Science Department, CUHK	Male
r_7	JeffreyYi	852	Jeffyi@abc.com	Computer Science Building, CUHK, Room 567B	M.
...	...	...	...	...	...

Positive-example set M : $\{RP_{1,6}, RP_{1,7}, RP_{2,5}, RP_{3,4}, RP_{6,7}\}$
Negative-example set D : $\{RP_{1,2}, RP_{1,3}, RP_{1,4}, RP_{1,5}, RP_{2,3}, RP_{2,4}, RP_{2,6}, RP_{2,7}, RP_{3,5}, RP_{3,6}, RP_{3,7}, RP_{4,5}, RP_{4,6}, RP_{4,7}, RP_{5,6}, RP_{5,7}\}$

Figure 1: A sample data of relation R ($RP_{i,j}$ denotes the record pair (r_i, r_j)).

to a high recall or a high precision), and it is hard to devise an adaptive algorithm for different preferences. Thirdly, as there are large numbers of record-matching rules and examples, and different record-matching rules are not independent, it is a big challenge to fully utilize these rules and examples. Fortunately, we have an observation that different similarity functions and thresholds have redundancy, and we have an opportunity to prune inappropriate similarity functions. To this end, we propose effective optimization techniques to eliminate such redundancy, and devise efficient algorithms to find the best similarity functions and thresholds. To summarize, we make the following contributions. (1) We formalize the problem of similarity-function identification in record-matching rules for entity matching. (2) We observe that different similarity functions and thresholds have redundancy, and devise efficient techniques to eliminate the redundancy. (3) We devise efficient algorithms to select the best similarity functions and thresholds. (4) We have conducted an extensive experiment study on both real and synthetic data sets. The experimental result shows that our algorithm achieves high accuracy and significantly outperforms the baseline algorithm.

Related Work. Entity Matching has been extensively studied in different communities [5,6,7,9,11,12,13,14,16,18, 20,21,22]. In the artificial intelligence community, machine-learning based techniques [5,9,20,21] usually learned a classifier (e.g., decision tree or SVM) from a given example set, and used the classifier to classify each record pair as matching or non-matching. However, these methods are rather expensive since they need to enumerate n^2 pairs. Although some heuristic techniques (e.g., blocking [4] and canopy clustering [19]) have been applied to filter "non-matching" pairs, they are at expense of decreasing the accuracy. In addition, these methods (e.g., SVM) usually employ black-box-based techniques and the results are not explainable.

As an alternative, the rule-based method [6,7,11,13,14,18] proposed from the database community has the advantages that the algorithm scales well and the obtained results are explainable. Typically, the rules can be specified based on domain knowledge. Recently, Fan et al. [11] proposed an effective algorithm for deducing record-matching rules from a small set of *matching dependencies*. As early mentioned, to accommodate errors in data, the rules may contain approximate-match conditions. However these methods neglect the problem of selecting optimal similarity functions and thresholds. It is challenging to select good similarity functions and thresholds, and inappropriate selections will lead to rather worse results (see Section 6.1). To address this problem, Chaudhuri et al. [6] proposed an operator tree for selecting similarity functions and thresholds. They modeled record-matching rules as a tree structure with the root node as a union operator, intermediate nodes as similarity-join operators and leaf nodes as relations. They identified the best operator tree based on a given set of positive and negative examples. However the method is rather inefficient for large numbers of similarity functions, since it does not consider the redundancy among similarity functions.

There were also many studies on similarity functions [17].

(a) ARs

λ_1^i : (name	, $\{f_e, f_g\}$	, [0,1])
λ_2^e : (tel	, $f_=$	, 1)
λ_3^e : (email	, $f_=$	, 1)
λ_4^i : (addr	, $\{f_e, f_j\}$	, [0,1])
λ_5^e : (gender	, $f_=$	, 1)

(b) RRs

ϕ_1	$\lambda_1^i \wedge \lambda_2^e$
ϕ_2	$\lambda_3^e \wedge \lambda_4^i$
ϕ_3	$\lambda_1^i \wedge \lambda_4^i \wedge \lambda_5^e$

Figure 2: Attribute-matching rules and record-matching rules ("$f_=$" denotes exact matching).

2. PROBLEM FORMULATION

Let $R[a_1, a_2, \cdots, a_n]$ be a relation with a set of attributes a_i ($i \in [1, n]$). Let r be a record in R and $r[a]$ be the value of the attribute a in record r. Given two records r and r', a similarity function $f(r[a], r'[a])$ computes a similarity score in $[0, 1]$. A larger score indicates $r[a]$ and $r'[a]$ have a higher similarity. As an example, we consider the following three similarity functions. (1) Edit Similarity f_e. The edit similarity between two strings s_1 and s_2 is defined as $1 - \frac{\mathsf{ed}(s_1,s_2)}{\max(|s_1|,|s_2|)}$, where $\mathsf{ed}(s_1, s_2)$ is the edit distance between s_1 and s_2, and $|s_1|(|s_2|)$ is the length of string $s_1(s_2)$. For example, given s_1 = "Jeffrey Yi" and s_2 = "Jeff*ery* Yi". $f_e(s_1, s_2) = 1 - \frac{2}{10} = 0.8$. (2) Jaccard Similarity f_j. Given two strings s_1 and s_2, we first tokenize them into two token sets $token(s_1)$ and $token(s_2)$. Their jaccard similarity is $\frac{|token(s_1) \cap token(s_2)|}{|token(s_1) \cup token(s_2)|}$. For example, $token(s_1)$={Jeffrey,Yi } and $token(s_2)$={Jeffery, Yi }. $f_j(s_1, s_2) = \frac{1}{3}$. (3) Gram-based Similarity f_g. Given two strings s_1 and s_2, we first generate their gram sets $gram(s_1)$ and $gram(s_2)$, where a q-gram is a substring with length q. For example, $gram(s_1)$ = {"Je", "ef", "ff", "fr", "re", "ey", "y ", " Y", "Yi"} ($q = 2$ in all running examples). The gram similarity is defined as $\frac{|gram(s_1) \cap gram(s_2)|}{|gram(s_1) \cup gram(s_2)|}$. For example, $f_g(s_1, s_2)$=$\frac{6}{12}$.

In order to decide whether two records r and r' in R refer to the same entity, we employ a matching-rule-based scheme to define their similarity. We below introduce a concept, "*explicit attribute-matching rule.*"

Definition 1. *An* explicit attribute-matching rule *(*eAR*) is a triple $\lambda^e(a, f, \theta)$, where a is an attribute name, f is a similarity function, and θ is a threshold. $r[a]$ and $r'[a]$ are considered to be the same, if $f(r[a], r'[a]) \geq \theta$. We use $(r[a], r'[a]) \vDash \lambda^e$ to denote that $r[a]$ and $r'[a]$ satisfy λ^e, $(r[a], r'[a]) \nvDash \lambda^e$ to denote that $r[a]$ and $r'[a]$ dissatisfy λ^e.*

The explicit attribute-matching rule includes an explicit similarity function and a threshold. We can use them to eas-

ily deduce whether two attribute values refer to the same entity. However in most cases users do not know how to define the similarly functions and how to determine the appropriate thresholds. Users may want to provide multiple similarity functions and a threshold range in the attribute-matching rule, and the system can find the best similarity function and a threshold in the range. We call such attribute-matching rules "*implicit attribute-matching rules*," defined as follows.

DEFINITION 2. *An* implicit attribute-matching rule *(*iAR*) is a triple $\lambda^i(a, \mathcal{F}, \Theta)$, where a is an attribute name, $\mathcal{F} = \{f_1, f_2, \cdots\}$ is a set of similarity functions, and Θ is a range. $r[a]$ and $r'[a]$ are considered to be the same if there exists a similarity function $f \in \mathcal{F}$ and $f(r[a], r'[a]) \geq \theta$ where θ is the lower bound of Θ.*

Given an iAR $(a, \mathcal{F}, \Theta)$ and an eAR (a, f, θ), the eAR is an instance of the iAR (or iAR is a generalization of the eAR) if $f \in \mathcal{F}$ and $\theta \in \Theta$. We can generate many instances of the iAR by enumerating the similarity functions in $\mathcal{F}$ and selecting a threshold in the threshold range.

For example, Figure 2(a) gives 2 iARs and 3 eARs. λ_1^i and λ_4^i are iARs. λ_2^e, λ_3^e, and λ_5^e are eARs. $\lambda^e : (name, f_e, 0.8)$ is an instance of λ_1^i. We use "$f_=$" to denote exact matching, that is $r[a]$ are similar to $r'[a]$ if and only if $r[a] = r'[a]$.

Based on these two concepts, we can define the concept of "*record-matching rule* (RR)".

DEFINITION 3. *A record-matching rule (*RR*) is a set of (explicit or implicit) attribute-matching rules, denoted by $\phi = \bigwedge_{i=1}^{k} \lambda_i$. A record pair (r, r') satisfies the record-matching rule, denoted by $(r, r') \vDash \phi$, if $(r[a], r'[a])$ satisfies every attribute-matching rule λ_i for $1 \leq i \leq k$. We also use $(r, r') \nvDash \phi$ to denote that (r, r') dissatisfies ϕ.*

A record-matching rule $\phi = \bigwedge_{i=1}^{k} \lambda_i$ is called an *explicit record-matching rule* (eRR) if every λ_i is an explicit attribute-matching rule for $1 \leq i \leq k$; otherwise it is called an implicit record-matching rule (iRR). For examples, Figure 2(b) shows 3 iRRs. Consider $\phi_1 = \lambda_1^i \wedge \lambda_2^e$ which indicates that two records refer to the same entity if two records have the *similar* names (where users want to select a similarity function from f_e and f_g, and the threshold can be any value in $[0, 1]$) and the same telephone number. To help readers better understand, we summarizes the notations used in this paper in Appendix A.

Traditional methods usually use explicit record-matching rules to quantify similarity of records. In this paper, we employ implicit record-matching rules to quantify the similarity and study how to select the *best* instances (eRRs) from a given set of iRRs. Note that iRRs can be gotten using existing methods, such as the schema-based methods [11].

Next we discuss how to evaluate the quality of an instance eRR. To address this problem, we suppose that users can give a set of examples, denoted by E, including positive examples, e.g., which records are known to be the same entity, denoted by M, and negative examples, e.g., which records are known to be not the same entity, denoted by D. Obviously $M \cup D = E$ and $M \cap D = \emptyset$.

Consider a set of eRRs Ψ. Given an eRR, $\psi \in \Psi$, let $\overline{M}_\psi = \{(r, r') | (r, r') \in E, (r, r') \vDash \psi\}$ be the set of record pairs generated by ψ (record pairs that satisfy ψ). Let $\overline{M}_\Psi = \bigcup_{\psi \in \Psi} \overline{M}_\psi$ be the set of record pairs generated by Ψ. Ideally, we hope that $\overline{M}_\Psi$ is exactly equal to M. However, in reality $\overline{M}_\Psi$ may contain negative record pairs. To evaluate the quality of Ψ, in this paper we focus on a general case of objective functions $F(\Psi, M, D)$: the larger $|\overline{M}_\Psi \cap M|$, the larger $F(\Psi, M, D)$; the smaller $|\overline{M}_\Psi \cap D|$, the larger $F(\Psi, M, D)$. Many functions belong to this general class. For example, the well-know F-measure function in information retrieval is a general objective function $\frac{2}{\frac{1}{\mathsf{p}} + \frac{1}{\mathsf{r}}}$, where $\mathsf{p} = \frac{|\overline{M}_\Psi \cap M|}{|\overline{M}_\Psi \cap M| + |\overline{M}_\Psi \cap D|}$ is the precision, and $\mathsf{r} = \frac{|\overline{M}_\Psi \cap M|}{|M|}$ is the recall. Users can tune the weights of precision or recall to select their preferences: preferring to recall or precision.

Now we are ready to formalize the problem of *si*milarity *f*unction *i*dentification in implicit record-matching rules for effective entity matching (called SiFi).

DEFINITION 4. *Given a set of* RRs *Φ, a set of positive examples M, and a set of negative examples D,* SiFi *finds a set of instances (*eRRs*) Ψ from Φ to maximize a pre-defined objective function $F(\Psi, M, D)$.*

Consider the relation R in Figure 1. The positive-example set $M = \{\text{RP}_{1,6}, \text{RP}_{1,7}, \cdots\}$ contains five record pairs and the negative-example set $D = \{\text{RP}_{1,2}, \text{RP}_{1,3}, \cdots\}$ contains 16 record pairs. Given a set of RRs in Figure 2, suppose the user-defined objective function is F-measure. SiFi finds a set Ψ of eRRs to maximize F-measure. Suppose we choose the similarity function f_e and the threshold 0.7 for λ_1^i, and choose the similarity function f_j and the threshold 0.8 for λ_4^i. Then the set of eRRs is $\Psi = \{\lambda_1^e \wedge \lambda_2^e,\ \lambda_3^e \wedge \lambda_4^e,\ \lambda_1^e \wedge \lambda_4^e \wedge \lambda_5^e\}$ where $\lambda_1^e = (name, f_e, 0.7)$ and $\lambda_4^e = (addr, f_j, 0.8)$. Since $f_e(r_1[name], r_7[name]) = 0.9 \geq 0.7$ and $r_1[tel] = r_7[tel]$, the record pair $\text{RP}_{1,7}$ satisfies $\lambda_1^e \wedge \lambda_2^e$. Similarly, we can also verify $\text{RP}_{1,5} \vDash \lambda_1^e \wedge \lambda_2^e$, $\text{RP}_{3,4} \vDash \lambda_3^e \wedge \lambda_4^e$ and $\text{RP}_{1,6} \vDash \lambda_1^e \wedge \lambda_4^e \wedge \lambda_5^e$. Therefore, $\overline{M}_\Psi = \{\text{RP}_{1,7}, \text{RP}_{1,5}, \text{RP}_{3,4}, \text{RP}_{1,6}\}$ where three record pairs $\text{RP}_{1,7}$, $\text{RP}_{3,4}$ and $\text{RP}_{1,6}$ are in M, and one record pair $\text{RP}_{1,5}$ is in D. The precision is $\mathsf{p} = \frac{|\overline{M}_\Psi \cap M|}{|\overline{M}_\Psi \cap M| + |\overline{M}_\Psi \cap D|} = \frac{3}{3+1}$ and the recall is $\mathsf{r} = \frac{|\overline{M}_\Psi \cap M|}{|M|} = \frac{3}{5}$. The objective-function value of Ψ is $\frac{2}{\frac{1}{\mathsf{p}} + \frac{1}{\mathsf{r}}} = \frac{2}{\frac{4}{3} + \frac{5}{3}} = \frac{2}{3}$. This value may not be the maximum one. In this paper we study how to efficiently find the *best* Ψ to maximize the value.

3. FROM INFINITE THRESHOLDS TO FINITE THRESHOLDS

A naive method to address SiFi problem is to enumerate all possible eRRs and select the best eRRs to maximize the objective function. However, as an iAR may contain a threshold range, there are infinite values. For example in Figure 2, we can get many eRRs from $\lambda_1^i \wedge \lambda_2^e$ as there are infinite values in the threshold range of λ_1^i ($\Theta = [0, 1]$). As we are only interested in those eRRs which can maximize the objective function, we want to generate finite eRRs which can also maximize the objective function.

Consider $\lambda^i(a, \mathcal{F}, \Theta)$ and its two instances, $\lambda_1^e(a, f, \theta_1)$ and $\lambda_2^e(a, f, \theta_2)$. Without loss of generality, suppose $\theta_1 < \theta_2$. Obviously the set of examples that satisfy λ_2^e is a subset of that of λ_1^e, that is, $\overline{M}_{\lambda_2^e} \subseteq \overline{M}_{\lambda_1^e}$ (where $\overline{M}_{\lambda^e}$ denotes the record pairs that satisfy λ^e). If there is no positive example in $\overline{M}_{\lambda_1^e} - \overline{M}_{\lambda_2^e}$, for any iRRs that contain λ^i, we will not use the eRRs with λ_1^e since they cannot get a better objective value than those with λ_2^e. The following Theorem shows the correctness of this idea (all the proofs are in Appendix B).

THEOREM 1. *Consider a set of* RRs *Φ, a set of positive examples M, a set of negative examples D, and an objective*

Table 1: Similarity for iAR λ_1^i : $(name, \{f_e, f_g\}, [0,1])$.

rid pairs	f_e	f_g	rid pairs	f_e	f_g	rid pairs	f_e	f_g
$RP_{1,2}$	0.4	0.23	$RP_{2,4}$	0.13	0	$RP_{3,7}$	0.8	0.78
$RP_{1,3}$	0.8	0.55	$RP_{2,5}$	**0.73**	**0.55**	$RP_{4,5}$	0.09	0
$RP_{1,4}$	0.1	0	$RP_{2,6}$	0.4	0.23	$RP_{4,6}$	0.1	0
$RP_{1,5}$	0.73	0.58	$RP_{2,7}$	0.44	0.25	$RP_{4,7}$	0	0
$RP_{1,6}$	**0.8**	**0.5**	$RP_{3,4}$	**0.1**	**0.09**	$RP_{5,6}$	0.55	0.27
$RP_{1,7}$	**0.9**	**0.7**	$RP_{3,5}$	0.64	0.5	$RP_{5,7}$	0.64	0.5
$RP_{2,3}$	0.4	0.25	$RP_{3,6}$	0.6	0.21	$RP_{6,7}$	**0.7**	**0.31**

function $F(\Psi, M, D)$. Consider an iRR *in Φ which contains an* iAR *$\lambda^i : (a, \mathcal{F}, \Theta)$, and two instances of λ^i, $\lambda_1^e : (a, f, \theta_1)$ and $\lambda_2^e(a, f, \theta_2)$. Suppose Ψ_1 is an* eRR *set, Ψ_2 is another* eRR *set transformed from Ψ_1 by replacing λ_1^e in Ψ_1 with λ_2^e. If $\theta_1 < \theta_2$ and there is no positive example in $\overline{M}_{\lambda_1^e} - \overline{M}_{\lambda_2^e}$, we have $F(\Psi_1, M, D) \leq F(\Psi_2, M, D)$.*

Based on Theorem 1, we have an observation that a large number of eRRs can be pruned since they cannot provide a better objective value. For example, consider an iAR $\lambda_1^i : (name, \{f_e, f_g\}, [0,1])$ in Figure 2. It has two similarity functions, edit similarity f_e and gram-based similarity f_g. Table 1 shows $f_e(r[name], r'[name])$ and $f_g(r[name], r'[name])$ of each record pair (r, r') in E. The positive examples are marked by the gray background color (e.g. $RP_{1,6}$). Consider two eARs $\lambda_1^e : (a, f_e, 0.6)$ and $\lambda_2^e : (a, f_e, 0.7)$, $\overline{M}_{\lambda_1^e} = \{RP_{1,3}, RP_{1,5}, RP_{1,6}, RP_{1,7}, RP_{2,5}, RP_{3,5}, RP_{3,6}, RP_{3,7}, RP_{5,7}, RP_{6,7}\}$, $\overline{M}_{\lambda_2^e} = \{RP_{1,3}, RP_{1,5}, RP_{1,6}, RP_{1,7}, RP_{2,5}, RP_{3,7}, RP_{6,7}\}$, $\overline{M}_{\lambda_1^e} - \overline{M}_{\lambda_2^e} = \{RP_{3,5}, RP_{3,6}, RP_{5,7}\}$. As there is no positive example in $\overline{M}_{\lambda_1^e} - \overline{M}_{\lambda_2^e}$, we can prune λ_1^e.

Using this idea, we can generate a finite set of eARs, and take them as a candidate eAR set, which can maximize the object function. Given an iAR $\lambda^i : (a, \mathcal{F}, \Theta)$, we construct a finite candidate eAR set of λ^i, denoted by $\mathcal{P}(\lambda^i)$, as follows. For each similarity function $f \in \mathcal{F}$, we add $\lambda^e : (a, f, \theta_{max})$ into $\mathcal{P}(\lambda^i)$ where θ_{max} is the upper bound of Θ (as it cannot be pruned due to it is the maximal value); for each positive example (r, r'), we compute $\theta = f(r[a], r'[a])$, if $\theta \in \Theta$, we add $\lambda^e : (a, f, \theta)$ into $\mathcal{P}(\lambda^i)$. For all other thresholds we can prune them as proved in Corollary 1.

COROLLARY 1. *Given an* iAR *$\lambda^i : (a, \mathcal{F}, \Theta)$ and a set of positive examples M, we can use the* candidate eAR set *$\mathcal{P}(\lambda^i)$ to generate the best* eRR *set.*

EXAMPLE 1. *Consider the relation in Figure 1. We show how the algorithm computes $\mathcal{P}(\lambda_1^i)$ for $\lambda_1^i : (name, \{f_e, f_g\}, [0,1])$. Firstly, for f_e and f_g we add two* eRRs *with the threshold $\theta_{max} = 1$ into $\mathcal{P}(\lambda_1^i) = \{(name, f_e, 1), (name, f_g, 1)\}$. Then we enumerate each record pair in $M : \{RP_{1,6}, RP_{1,7}, RP_{2,5}, RP_{3,4}, RP_{6,7}\}$. For the first record pair $RP_{1,6}$, we compute the similarity f_e("Jeffrey Yi", "Jeffery Yi") = 0.8 and f_g("Jeffrey Yi", "Jeffery Yi") = 0.5 as shown in Table 1, and then add $(name, f_e, 0.8)$ and $(name, f_g, 0.5)$ into $\mathcal{P}(\lambda_1^i)$. We repeat these steps for the rest of record pairs in M. Finally, we get $\mathcal{P}(\lambda_1^i)$ with 12* eARs *as shown in Table 2.*

4. ELIMINATING REDUNDANCY

The size of candidate eAR set generated in Section 3 may be quite large. In this section, we show that a large number of eARs, called redundant eARs, can not lead to an optimal objective-function value. In other words, we can eliminate such redundancy to reduce the size of candidate eAR set.

Table 2: Candidate eAR Set of λ_1^i: $(name, \{f_e, f_g\}, [0,1])$.

λ_1^e	(name,f_e, 1)	λ_7^e	(name,f_g, 1)
λ_2^e	(name,f_e, 0.9)	λ_8^e	(name,f_g, 0.7)
λ_3^e	(name,f_e, 0.8)	λ_9^e	(name,f_g, 0.55)
λ_4^e	(name,f_e, 0.73)	λ_{10}^e	(name,f_g, 0.5)
λ_5^e	(name,f_e, 0.7)	λ_{11}^e	(name,f_g, 0.31)
λ_6^e	(name,f_e, 0.1)	λ_{12}^e	(name,f_g, 0.09)

Given a candidate eAR set $\mathcal{P}(\lambda^i)$, we first divide it into $|\mathcal{F}|$ groups, where $|\mathcal{F}|$ is the size of λ^i's similarity function set $\mathcal{F}$. Let $\mathcal{G}_f$ denote a group of candidate eARs in $\mathcal{P}(\lambda^i)$ whose similarity function is f. For simplicity, in one group we say $\lambda_1^e > \lambda_2^e$ iff λ_1^e has a larger threshold than λ_2^e. λ^e is maximal (minimal) means it has the maximal (minimal) threshold. For example, in Table 2 λ_1^i has two similarity functions. $\mathcal{P}(\lambda_1^i)$ is divided into two groups $\mathcal{G}_{f_e} = \{\lambda_1^e, \lambda_2^e, \cdots, \lambda_6^e\}$ and $\mathcal{G}_{f_g} = \{\lambda_7^e, \lambda_8^e, \cdots, \lambda_{12}^e\}$. We say $\lambda_3^e > \lambda_4^e$ since the threshold of λ_3^e (0.8) is larger than that of λ_4^e (0.73). Note that we cannot say $\lambda_3^e > \lambda_9^e$ since they are not in the same group. We observe that in each group there may exist threshold redundancy (Section 4.1), and between two different groups there may exist similarity-function redundancy (Section 4.2).

4.1 Threshold Redundancy

Given an iAR $\lambda^i : (a, \mathcal{F}, \Theta)$, there may be still many eARs of λ^i in $\mathcal{P}(\lambda^i)$, we can remove some of them based on the following observation. Consider two eARs $\lambda_1^e : (a, f, \theta_1)$ and $\lambda_2^e : (a, f, \theta_2)$ in $\mathcal{P}(\lambda^i)$. Suppose $\theta_1 > \theta_2$, then $\overline{M}_{\lambda_1^e} \subset \overline{M}_{\lambda_2^e}$ (If $\overline{M}_{\lambda_1^e} = \overline{M}_{\lambda_2^e}$, λ_2^e will be removed based on Theorem 1). Obviously, λ_1^e and λ_2^e generate the same record pairs $\overline{M}_{\lambda_1^e}$, and λ_2^e can identify more record pairs $\overline{M}_{\lambda_2^e} - \overline{M}_{\lambda_1^e}$. If these pairs are all positive examples, that is $\overline{M}_{\lambda_2^e} - \overline{M}_{\lambda_1^e} \subseteq M$, then λ_2^e will be better than λ_1^e. We say λ_1^e is redundant w.r.t λ_2^e. The correctness is proved in Theorem 2.

THEOREM 2. *Given a* candidate eAR set *$\mathcal{P}(\lambda^i)$ and a set M of positive examples, $\lambda_1^e : (a, f, \theta_1) \in \mathcal{P}(\lambda^i)$ is redundant w.r.t $\lambda_2^e : (a, f, \theta_2) \in \mathcal{P}(\lambda^i)$ if $\overline{M}_{\lambda_2^e} - \overline{M}_{\lambda_1^e} \subseteq M$ and $\theta_1 > \theta_2$.*

A simple algorithm to detect such redundancy is to enumerate $|\mathcal{F}| \cdot |\mathcal{G}_f|$ candidate eARs, and takes $\mathcal{O}(|\mathcal{G}_f| \cdot |E|)$ to verify the redundancy of each candidate. The algorithm needs $\mathcal{O}(|\mathcal{F}| \cdot |\mathcal{G}_f|^2 \cdot |E|)$ time. Next we propose an efficient algorithm that reduces the time complexity to $\mathcal{O}(|\mathcal{F}| \cdot |K| \cdot |E|)$ where $|K| = log(|\mathcal{F}| \cdot |\mathcal{G}_f|)$ (See analysis in Appendix C).

We use a compressed inverted index CIX to detect threshold redundancy. An inverted index over $\mathcal{P}(\lambda^i)$ maps each $\lambda^e \in \mathcal{P}(\lambda^i)$ to a list of record pairs that satisfy λ^e (i.e. $\overline{M}_{\lambda^e}$). We can compress the inverted index since if $\lambda_1^e > \lambda_2^e$, $\overline{M}_{\lambda_1^e} \subseteq \overline{M}_{\lambda_2^e}$. Suppose λ_1^e is the maximal eAR and λ_2^e is the second maximal eAR in $\mathcal{P}(\lambda_1^i)$. The compressed inverted list $CIX(\lambda_2^e)$ of λ_2^e only stores the record pairs in $\overline{M}_{\lambda_2^e} - \overline{M}_{\lambda_1^e}$. Iteratively we construct the compressed inverted index for all eARs. Figure 3 shows the compressed inverted index over the two groups of $\mathcal{P}(\lambda_1^i)$. For example, $CIX(\lambda_3^e) = \overline{M}_{\lambda_3^e} - \overline{M}_{\lambda_2^e} = \{RP_{1,6}, RP_{1,3}, RP_{3,7}\}$. We partition CIX into two lists: $CIX_M(\lambda^e)$ and $CIX_D(\lambda^e)$, which respectively denote the positive and the negative record pairs in $CIX(\lambda^e)$. For example, $CIX_M(\lambda_3^e) = \{RP_{1,6}\}$ and $CIX_D(\lambda_3^e) = \{RP_{1,3}, RP_{3,7}\}$.

Recall Theorem 2, to verify whether λ_1^e is redundant, we need to check $\overline{M}_{\lambda_2^e} - \overline{M}_{\lambda_1^e} \subseteq M$ for every $\lambda_2^e \in \mathcal{P}(\lambda^i)$ that is smaller than λ_1^e. However, note that we only need to check the maximal eAR λ_2^e such that $\lambda_2^e < \lambda_1^e$, since if λ_1^e

Candidate eARs	Compressed Inverted List	
	M	D
λ_1^e	-	-
λ_2^e	$\text{RP}_{1,7}$	-
λ_3^e	$\text{RP}_{1,6}$	$\text{RP}_{1,3}$,$\text{RP}_{3,7}$
λ_4^e	$\text{RP}_{2,5}$	$\text{RP}_{1,5}$
λ_5^e	$\text{RP}_{6,7}$	-
λ_6^e	$\text{RP}_{3,4}$	$\text{RP}_{1,2}$,$\text{RP}_{1,4}$,$\text{RP}_{2,3}$,$\text{RP}_{2,4}$,$\text{RP}_{2,6}$,$\text{RP}_{2,7}$, $\text{RP}_{3,5}$,$\text{RP}_{3,6}$,$\text{RP}_{4,6}$,$\text{RP}_{5,6}$,$\text{RP}_{5,7}$
λ_7^e	-	-
λ_8^e	$\text{RP}_{1,7}$	$\text{RP}_{3,7}$
λ_9^e	$\text{RP}_{2,5}$	$\text{RP}_{1,3}$,$\text{RP}_{1,5}$
λ_{10}^e	$\text{RP}_{1,6}$	$\text{RP}_{3,5}$,$\text{RP}_{5,7}$
λ_{11}^e	$\text{RP}_{6,7}$	-
λ_{12}^e	$\text{RP}_{3,4}$	$\text{RP}_{1,2}$,$\text{RP}_{2,3}$,$\text{RP}_{2,6}$,$\text{RP}_{2,7}$,$\text{RP}_{3,6}$,$\text{RP}_{5,6}$

Figure 3: A compressed inverted index for $\mathcal{P}(\lambda_1^i)$.

is not redundant w.r.t λ_2^e (i.e. $\overline{M}_{\lambda_2^e} - \overline{M}_{\lambda_1^e} \nsubseteq M$), then for any smaller eAR $\lambda_3^e < \lambda_2^e$, we have $\overline{M}_{\lambda_2^e} \subseteq \overline{M}_{\lambda_3^e}$, thus $\overline{M}_{\lambda_3^e} - \overline{M}_{\lambda_1^e} \nsubseteq M$. To check whether $\overline{M}_{\lambda_2^e} - \overline{M}_{\lambda_1^e} \subseteq M$, we can use the compressed inverted index to verify this condition. Since $CIX(\lambda_2^e) = \overline{M}_{\lambda_2^e} - \overline{M}_{\lambda_1^e}$, we only need to check the condition $CIX(\lambda_2^e) \subseteq M$, that is whether $CIX_D(\lambda_2^e)$ is empty. For example, consider the compressed inverted index in Figure 3. We can see λ_1^e, λ_4^e, λ_{10}^e are redundant since $CIX_D(\lambda_2^e)$, $CIX_D(\lambda_5^e)$, $CIX_D(\lambda_{11}^e)$ are empty, respectively.

4.2 Similarity-Function Redundancy

In this section, we study redundancy among different similarity functions. We observe that 1) Similarity functions tend to return a higher score to a similar pair and a lower score to a dissimilar pair, and they may give nearly the same scores for some similar pairs; 2) Similarity functions may share a common inherent coherence. For example, consider some token-based similarity functions such as Jaccard and Cosine. They all share an inherent coherence: if two strings share many tokens, any token-based similarity functions will return a higher value [7]. Based on these observations, we discuss how to find redundancy among similarity functions.

Consider two eARs $\lambda_1^e : (a, f_1, \theta_1)$ and $\lambda_2^e : (a, f_2, \theta_2)$ in $\mathcal{P}(\lambda^i)$ where $f_1 \neq f_2$. We first consider a special case such that, for each record pair, λ_1^e and λ_2^e always generate the same record pairs, i.e. $\overline{M}_{\lambda_1^e} = \overline{M}_{\lambda_2^e}$. From the viewpoint of RR set, they have no difference and we can only keep one of them. To make this observation more general, we find that 1) for each positive pair (r, r') (i.e. $(r, r') \in M$), if λ_1^e takes this pair as similar, λ_2^e will also takes it as similar, and 2) for each negative pair (r, r') (i.e. $(r, r') \in D$), if λ_1^e takes this pair as dissimilar, λ_2^e will also takes it as dissimilar, then λ_1^e is redundant w.r.t λ_2^e since it can not lead to a better objective value than λ_2^e. The first statement implies $\overline{M}_{\lambda_1^e} \cap M \subseteq \overline{M}_{\lambda_2^e} \cap M$, and the second statement is equivalent to $\overline{M}_{\lambda_1^e} \cap D \supseteq \overline{M}_{\lambda_2^e} \cap D$. Theorem 3 shows the correctness.

THEOREM 3. *Given a* candidate eAR set $\mathcal{P}(\lambda^i)$, *a set M of positive examples and a set D of negative examples, $\lambda_1^e : (a, f_1, \theta_1) \in \mathcal{P}(\lambda^i)$ is redundant w.r.t $\lambda_2^e : (a, f_2, \theta_2) \in \mathcal{P}(\lambda^i)$ if $\overline{M}_{\lambda_1^e} \cap M \subseteq \overline{M}_{\lambda_2^e} \cap M$ and $\overline{M}_{\lambda_1^e} \cap D \supseteq \overline{M}_{\lambda_2^e} \cap D$.*

A simple algorithm to detect such redundancy is to enumerate $|\mathcal{P}(\lambda^i)|$ candidate eARs, and takes $\mathcal{O}\big(|\mathcal{P}(\lambda^i)| \cdot |E|\big)$ to verify the redundancy of each candidate. The algorithm needs $\mathcal{O}\big(|\mathcal{P}(\lambda^i)|^2 \cdot |E|\big)$ time. It is expensive since $|\mathcal{P}(\lambda^i)|$ may be quite large. Next we propose an efficient algorithm that reduces the time complexity to $\mathcal{O}\big(|\mathcal{F}| \cdot |K| \cdot |E|\big)$ where $|K| = \min\Big(|\mathcal{F}|, log(|\mathcal{P}(\lambda^i)|)\Big)$ (See analysis in Appendix C).

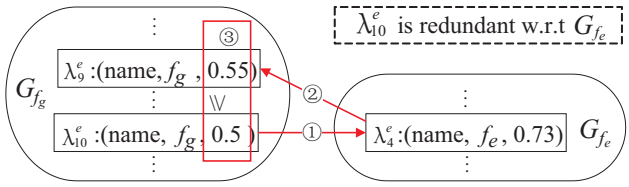


Figure 4: Verify similarity-function redundancy.

The basic idea is as follows. Consider two groups $\mathcal{G}_{f_1}$, $\mathcal{G}_{f_2}$ with similarity functions f_1, f_2. Since similarity-function redundancy exists between two different groups, we only need to study how to detect redundant eARs in $\mathcal{G}_{f_1}$ w.r.t $\mathcal{G}_{f_2}$. Consider an eAR $\lambda_1^e \in \mathcal{G}_{f_1}$. Next we show for each eAR $\lambda_1^e \in \mathcal{G}_{f_1}$ we only need $\mathcal{O}(1)$ time to verify whether λ_1^e is redundant w.r.t $\mathcal{G}_{f_2}$. Let λ_M^e be the maximal eAR in $\mathcal{G}_{f_2}$ such that $\overline{M}_{\lambda_1^e} \cap M \subseteq \overline{M}_{\lambda_M^e} \cap M$. For other eARs $\lambda^e \in \mathcal{G}_{f_2}$, if $\lambda^e > \lambda_M^e$, since λ_M^e is maximal, $\overline{M}_{\lambda_1^e} \cap M \nsubseteq \overline{M}_{\lambda^e} \cap M$; if $\lambda^e \leq \lambda_M^e$, then $\overline{M}_{\lambda_M^e} \subseteq \overline{M}_{\lambda^e}$, thus $\overline{M}_{\lambda_1^e} \cap M \subseteq \overline{M}_{\lambda^e} \cap M$. Therefore, to validate whether λ_1^e is redundant w.r.t $\mathcal{G}_{f_2}$, we only need to check if there exists an eAR λ^e in $C = \{\lambda^e \in \mathcal{G}_{f_2} | \lambda^e \leq \lambda_M^e\}$ such that $\overline{M}_{\lambda_1^e} \cap D \supseteq \overline{M}_{\lambda^e} \cap D$. Since λ_M^e is the maximal eAR in C, we have $\overline{M}_{\lambda_M^e} \cap D \subseteq \overline{M}_{\lambda^e} \cap D$ for other λ^e in C, thus we only need to check $\overline{M}_{\lambda_1^e} \cap D \supseteq \overline{M}_{\lambda_M^e} \cap D$. Let λ_D^e be the maximal eAR in $\mathcal{G}_{f_1}$ such that $\overline{M}_{\lambda_D^e} \cap D \supseteq \overline{M}_{\lambda_M^e} \cap D$. If $\lambda_1^e > \lambda_D^e$, since λ_D^e is maximal, $\overline{M}_{\lambda_1^e} \cap D \ngeq \overline{M}_{\lambda_M^e} \cap D$; if $\lambda_1^e \leq \lambda_D^e$, since $\overline{M}_{\lambda_1^e} \supseteq \overline{M}_{\lambda_D^e}$, $\overline{M}_{\lambda_1^e} \cap D \supseteq \overline{M}_{\lambda_M^e} \cap D$. Therefore, λ_1^e is redundant w.r.t $\mathcal{G}_{f_2}$ if and only if $\lambda_1^e \leq \lambda_D^e$. Note that we can pre-compute λ_M^e and λ_D^e, and store them into two tables in advance (See details in Appendix D). Then given λ^e, we can get λ_M^e and λ_D^e with $\mathcal{O}(1)$ time. Example 2 shows how this idea works.

EXAMPLE 2. *In Figure 4, consider $\lambda_{10}^e : (name, f_g, 0.5)$ in $\mathcal{G}_{f_g}$. We want to check whether λ_{10}^e is redundant w.r.t $\mathcal{G}_{f_e}$. Firstly, we find the maximal λ_M^e in $\mathcal{G}_{f_e}$ such that $\overline{M}_{\lambda_{10}^e} \cap M \subseteq \overline{M}_{\lambda_M^e} \cap M$. From Table 1, we can compute $\overline{M}_{\lambda_{10}^e} \cap M = \{\text{RP}_{1,6}, \text{RP}_{1,7}, \text{RP}_{2,5}\}$. Since $\overline{M}_{\lambda_{10}^e} \cap M \subseteq \overline{M}_{\lambda_4^e} \cap M = \{\text{RP}_{1,6}, \text{RP}_{1,7}, \text{RP}_{2,5}\}$ and $\overline{M}_{\lambda_{10}^e} \cap M \nsubseteq \overline{M}_{\lambda_3^e} \cap M = \{\text{RP}_{1,6}, \text{RP}_{1,7}\}$, the maximal λ_M^e is λ_4^e:$(name, f_e, 0.73)$. Secondly, we find the maximal λ_D^e in $\mathcal{G}_{f_g}$ such that $\overline{M}_{\lambda_4^e} \cap D \subseteq \overline{M}_{\lambda_D^e} \cap D$. From Table 1, we can compute $\overline{M}_{\lambda_4^e} \cap D = \{\text{RP}_{1,3}, \text{RP}_{1,5}, \text{RP}_{3,7}\}$. Since $\overline{M}_{\lambda_4^e} \cap D \subseteq \overline{M}_{\lambda_9^e} \cap D = \{\text{RP}_{1,3}, \text{RP}_{1,5}, \text{RP}_{3,7}\}$ and $\overline{M}_{\lambda_4^e} \cap D \nsubseteq \overline{M}_{\lambda_8^e} \cap M = \{\text{RP}_{1,3}\}$, the maximal λ_D^e is $\lambda_9^e : (name, f_g, 0.55)$. Thirdly, we compare λ_{10}^e with λ_9^e. Since $\lambda_{10}^e \leq \lambda_9^e$, λ_{10}^e is redundant w.r.t $\mathcal{G}_{f_e}$.*

Eliminating Redundancy: Our method has two salient features. (1) Eliminating threshold redundancy has no effect on similarity-function redundancy. Therefore, for a candidate eAR set, we can first eliminate its threshold redundancy, then eliminate its similarity-function redundancy. (2) The eliminating order of function redundancy and threshold redundancy will not change the final results. That is we get the same results for the two cases (a) eliminating function redundancy first and then threshold redundancy; and (b) eliminating threshold redundancy first and then function redundancy. (Appendix E shows the correctness).

5. ALGORITHMS FOR SiFi PROBLEM

A brute-force algorithm to solve the SiFi problem is to first enumerate all candidates and then select the one with

the maximal value as follows. Given a set of RRs Φ, we first generate $\mathcal{P}(\lambda^i)$ for each iAR λ^i in Φ and reduce $\mathcal{P}(\lambda^i)$ to $\mathcal{P}^n(\lambda^i)$ by eliminating redundancy. Then we enumerate all the combinations of eRRs and the number of candidates is $\prod_{\lambda^i \in \Phi} |\mathcal{P}^n(\lambda^i)|$. For each candidate, we compute its objective-function value, and select the one with the maximal value. However as there are large numbers of candidates, this method is very expensive. Suppose there are 6 RRs, each RR contains 2 iARs, and there are 5 functions and 10 possible thresholds. Then there will be 50^{12} candidates.

In addition, the SiFi problem is *NP-hard* as formalized in Theorem 4, which can be proved using a reduction from the maximum-coverage problem [15]. We propose three heuristic algorithms SiFi-Greedy, SiFi-Gradient and SiFi-Hill.

THEOREM 4. *The* SiFi *problem is NP-hard.*

SiFi-Greedy: We propose SiFi-Greedy inspired by the greedy algorithm for maximum-coverage problem [15]. Intuitively, for a given set Φ of RRs, SiFi-Greedy picks the *best* iAR in Φ at each time and terminates until all the iARs are picked. Specifically, the algorithm first evaluates the *quality* of each iAR in Φ based on the example set E, and then picks the iAR with the highest quality, denoted by λ^i_{max}. Next it updates Φ by changing λ^i_{max} to the eAR with the highest quality in $\mathcal{P}^n(\lambda^i_{max})$, and updates the example set E by removing the examples that satisfy the eRRs in Φ. The algorithm terminates when there is no iAR in Φ. The *quality* of an eAR λ^e w.r.t $E = M \cup D$ is defined as $\mathcal{Q}(\lambda^e) = F(\lambda^e, M, D)$ where F is the same objective function used to quantity eRRs (λ^e is taken as a special eRR with a single eAR). For an iAR λ^i, since any eAR in $\mathcal{P}^n(\lambda^i)$ can become its instance, we define its quality $\mathcal{Q}(\lambda^i)$ as the maximum of $\mathcal{Q}(\lambda^e)$ for $\lambda^e \in \mathcal{P}^n(\lambda^i)$. Example 3 shows how SiFi-Greedy works.

EXAMPLE 3. *Suppose the objective function* $F(\Psi, M, D) = |\overline{M}_\Psi \cap M| - 0.5 \cdot |\overline{M}_\Psi \cap D|$. *In our running example, there are two* iARs λ^i_1 *and* λ^i_4, *thus SiFi-Greedy needs two steps. At the first step, we first compute the quality of* λ^i_1 *and* λ^i_4. *For* λ^i_1, *we obtain* $\mathcal{P}^n(\lambda^i_1) = \{\lambda^e_2, \lambda^e_3, \lambda^e_5, \lambda^e_{12}\}$ *after eliminating redundancy from the* candidate eAR set *in Table 2. As only* $\text{RP}_{1,7}$ *satisfies* λ^e_2, $\overline{M}_{\lambda^e_2} = \{\text{RP}_{1,7}\}$. *As* $\text{RP}_{1,7}$ *is in* M, $\overline{M}_{\lambda^e_2} \cap M = \{\text{RP}_{1,7}\}$ *and* $\overline{M}_{\lambda^e_2} \cap D = \{\}$. *Therefore,* $\mathcal{Q}(\lambda^e_2) = |\overline{M}_{\lambda^e_2} \cap M| - 0.5 \cdot |\overline{M}_{\lambda^e_2} \cap D| = 1$. *Similarly,* $\mathcal{Q}(\lambda^e_3) = 1$, $\mathcal{Q}(\lambda^e_5) = 2.5$ *and* $\mathcal{Q}(\lambda^e_{12}) = 0.5$. *Since* $\mathcal{Q}(\lambda^e_5)$ *is maximum,* $\mathcal{Q}(\lambda^i_1) = 2.5$. *Using the same method, we can compute* $\mathcal{Q}(\lambda^i_4) = 3.5$. *As* $\mathcal{Q}(\lambda^i_4) > \mathcal{Q}(\lambda^i_1)$, $\lambda^i_{max} = \lambda^i_4$. *Let* $\lambda^{e'}_4$ *denote the* eAR *with the highest quality in* $\mathcal{P}^n(\lambda^i_4)$. *Thus,* $\lambda^{e'}_4$ *is the instance of* λ^i_4. *We update* Φ *by changing* λ^i_4 *to* $\lambda^{e'}_4$, *and update the example set by removing the examples that satisfy the* eRRs *in* Φ, *i.e.* $\lambda^e_3 \wedge \lambda^{e'}_4$. *At the second step, based on the new example set, we recompute* $\mathcal{Q}(\lambda^i_1)$ *and update* Φ *by changing* λ^i_1 *to the* eAR *with the highest quality in* $\mathcal{P}^n(\lambda^i_1)$. *Finally, we obtain the instances of* λ^i_1 *and* λ^i_4.

SiFi-Gradient: The major problem of SiFi-Greedy is that iARs are optimized independently, and the interaction among different iARs is neglected. For example, in Figure 2, due to the occurrence of both λ^i_1 and λ^i_4 in ϕ_3, the best instance of λ^i_1 may change for different instances of λ^i_4, and vice versa. To address this problem, we devise an iterative algorithm, namely SiFi-Gradient, which is based on the idea of gradient descent. Intuitively, SiFi-Gradient iteratively adjusts the instances of iARs for reaching a higher objective value. Initially, given a set Φ of RRs, for each $\lambda^i \in \Phi$, the algorithm takes $\lambda^e \in \mathcal{P}^n(\lambda^i)$ with the highest $\mathcal{Q}(\lambda^e)$ as λ^i's instance, and computes the objective value of the corresponding eRRs. At each iteration, the algorithm changes the instance of each iAR to one of its *neighbors* which results in the largest objective value. SiFi-Gradient terminates the iteration when the objective value can not be larger. Consider the instance $\lambda^e : (a, f, \theta)$ of an iAR λ^i. Intuitively, the neighbors of λ^e consists of eARs whose thresholds are close to θ. Formally, we say $\lambda^e_1 : (a, f_1, \theta_1)$ is λ^e's neighbor if and only if $\lambda^e_1 \in \mathcal{P}^n(\lambda^i)$ and there does not exist $\lambda^e_2 : (a, f_1, \theta_2) \in \mathcal{P}^n(\lambda^i)$ such that $\theta_2 \in (\theta, \theta_1)$ or $\theta_2 \in (\theta_1, \theta)$. Example 4 shows how SiFi-Gradient works.

EXAMPLE 4. *Recall Example 3,* λ^e_5 *and* $\lambda^{e'}_4$ *are the* eARs *with the highest quality in* $\mathcal{P}^n(\lambda^i_1)$ *and* $\mathcal{P}^n(\lambda^i_4)$, *respectively. Initially, SiFi-Gradient takes* λ^e_5 *as* λ^i_1*'s instance and* $\lambda^{e'}_4$ *as* λ^i_4*'s instance, and computes the corresponding objective value, denoted as* op. *Next we iteratively adjust instances of* λ^i_1 *and* λ^i_4. *Consider the current instance* $\lambda^e_5 : (name, f_e, 0.7)$ *of* λ^i_1. *We first identify the neighbors of* λ^e_5 *from* $\mathcal{P}^n(\lambda^i_1) = \{\lambda^e_2, \lambda^e_3, \lambda^e_5, \lambda^e_{12}\}$. *We can see* λ^e_3 *is a neighbor of* λ^e_5 *since for other* eARs *with the same similarity function as* λ^e_3, *i.e.* $\lambda^e_2 : (name, f_e, 0.9)$, *the threshold of* λ^e_2 *is* $0.9 \notin (0.7, 0.8)$. *Similarly, we can identify another neighbor* λ^e_{12}. *We change* λ^e_5 *to one of its neighbors,* λ^e_3 *or* λ^e_{12} *and choose the neighbor, which can result in the largest objective value, as the new instance of* λ^i_1. *Using the similar method, we can identify a new instance for* λ^i_4. *SiFi-Gradient changes* λ^e_5 *and* $\lambda^{e'}_4$ *to new instances, and updates* op *to the new objective value, then goes to the next iteration. If the new objective value can not be larger than* op, *SiFi-Gradient terminates the iteration and returns the current instances of* λ^i_1 *and* λ^i_4.

SiFi-Hill: SiFi-Gradient terminates the iteration when the objective value can not be larger by changing current instances to their neighbors. In each iteration it only considers neighbors of current instances, and it only uses a subset of $\mathcal{P}^n(\lambda^i)$ and may lead to local optimal solution. The objective value may become larger by changing current instances to other eARs. To address this problem, we devise another iterative algorithm, namely SiFi-Hill, which is based on the idea of hill climbing. SiFi-Hill uses the same method as SiFi-Gradient to initialize the instances of iARs. At each iteration, different from SiFi-Gradient, the algorithm adjusts the instance of a single iAR and allows to change the instance of λ^i to any eAR in $\mathcal{P}^n(\lambda^i)$. SiFi-Hill terminates when the objective value can not be larger. Example 5 shows how SiFi-Hill works.

EXAMPLE 5. *Recall Example 4, initially SiFi-Hill takes* λ^e_5 *as* λ^i_1*'s instance and* $\lambda^{e'}_4$ *as* λ^i_4*'s instance, and computes the objective value* op. *Next we iteratively adjust instances of* λ^i_1 *and* λ^i_4. *We first fix* λ^e_5 *and try to replace* $\lambda^{e'}_4$ *with the other* eAR *in* $\mathcal{P}^n(\lambda^i_4)$, *then fix* $\lambda^{e'}_4$ *and try to replace* λ^e_5 *with the other* eAR *in* $\mathcal{P}^n(\lambda^i_1)$. *SiFi-Hill chooses the case that results in the largest objective value, and updates* op *to the new objective value, then goes to the next iteration. If the objective value can not be larger than* op, *SiFi-Hill terminates and returns the current instances of* λ^i_1 *and* λ^i_4.

We also explore the practical applicability of our approach to support missing attribute values, combination of similarity values and majority votes in Appendix F and provide the complexity analysis of three algorithms in Appendix G.

6. EXPERIMENTS

We have conducted experiment evaluation on both real and synthetic data sets: *Cora*, Restaurant and *DBGen*. We compared with the baseline methods and state-of-the-art methods, OpTrees [6] and SVM [5]. Appendix H.1 gives detailed data set descriptions and experimental settings.

6.1 Comparison with Baseline Methods

We compared SiFi-Gradient and SiFi-Hill with baseline methods SiFi-Greedy, SiFi-Equal, SiFi-Expert, where SiFi-Equal uses $f_=$ for each iAR and SiFi-Expert used the eARs formulated by experts (see Table 5 in Appendix H.1). We asked for three experts to formulate eARs who are familiar with the datasets, SiFi-Expert-1, SiFi-Expert-2 and SiFi-Expert-3. We used K-fold cross-validation to evaluate all methods.

We compared these methods on three data sets where the objective function is F-measure (We evaluated different objective functions in Appendix H.2). Figure 5 reports F-measure values by varying the number of folds (K). We can see SiFi-Hill and SiFi-Gradient outperform the baseline methods on two real data sets *Cora* and *Restaurant*. For example, in Figure 5(a) the values of SiFi-Hill are around 0.9 while the best baseline method SiFi-Greedy is around 0.7. This is because iARs are interdependent, and SiFi-Greedy neglects the interaction among different iARs. But for *DBGen*, SiFi-Greedy almost got the same objective value as SiFi-Hill since the data set contains errors in different attributes independently. SiFi-Hill performs better than SiFi-Gradient on *Cora* since SiFi-Gradient only enumerates neighbors at each iteration while SiFi-Gradient enumerates all possible eARs.

From the performance of SiFi-Equal, we can see it is important to match some attribute values approximately. For example, in Figure 5(a) the values of SiFi-Equal are below 0.1. The performance of SiFi-Expert shows the necessity of studying the SiFi problem. Firstly, experts can not select appropriate similarity functions and thresholds for iARs. For example, in Figure 5(a) the values of SiFi-Expert-1, SiFi-Expert-2, SiFi-Expert-3 can not even reach 0.5 on *Cora* data set. Secondly, it is hard for human to tell the difference of similarity functions and thresholds, so the eARs formulated by experts may lead to different results. For example, in Figure 5(b) the F-measure of SiFi-Expert-3 is around 0.8 while that of SiFi-Expert-1 are only around 0.6.

6.2 Evaluation of Eliminating Redundancy

We first compared the efficiency of eliminating redundancy on three data sets. In Figure 6, ER denotes the simple algorithm and ER-Op denotes the optimized algorithm in Section 4. Note that ER-Op contains the time of precomputing λ^e_M and λ^e_D. We can see ER-Op is faster than ER by several orders of magnitude. For example, in Figure 6(a) when the number of record pairs is 10^5, ER took 8492 seconds to eliminate redundancy, but ER-Op only took 5.5 seconds. Next we evaluated the effect of eliminating redundancy on SiFi-Hill. In Figure 6, SiFi-Hill, SiFi-Hill+ER and SiFi-Hill+ER-Op respectively denote the SiFi-Hill algorithm without eliminating redundancy, using ER algorithm to eliminate redundancy and using optimized ER algorithm to eliminate redundancy. We can see SiFi-Hill+ER-Op performs the best among the three algorithms. In Figure 6(c) for $5K$ record pairs, SiFi-Hill took 53.6 seconds and SiFi-Hill+ER took 121.1 seconds but SiFi-Hill+ER-Op only took 14.3 seconds. Figure 6 also illustrates two important findings. The first one is eliminating redundancy can improve the performance of SiFi-Hill. From Figure 6(a)-(c), we can see SiFi-Hill+ER-Op always outperforms SiFi-Hill. The second one is that optimizing the eliminating-redundancy algorithm is quite necessary. In Figure 6(a), SiFi-Hill+ER performs much worse than SiFi-Hill as ER is inefficient.

6.3 Comparison with existing techniques

We compared our methods with OpTrees and SVM for record matching. OpTrees is another efficient and explainable technique for record matching [6]. It needs a set of positive and negative examples to construct an executable operator tree and also three input parameters β, d and K, where β is used to adjust the precision, d denotes the maximal number of similarity functions in all RRs and K denotes the maximal number of RRs in the operator tree. In the experiment, we set β to the value such that the precision is no larger than 0.9, $d = 2$[1] and $K = 4$. SVM is a machine-learning technique for record matching that has been shown to significantly outperform other machine-learning techniques such as decision trees [5]. Given a data set with n attributes, we represent each record pair as a $n|\mathcal{F}|$-dimensional vector where each component denotes similarity between two attribute values of the record that is calculated using one of the $|\mathcal{F}|$ similarity functions. We implemented OpTrees by ourselves and obtained the implementation of SVM from Bilenko et al [5].

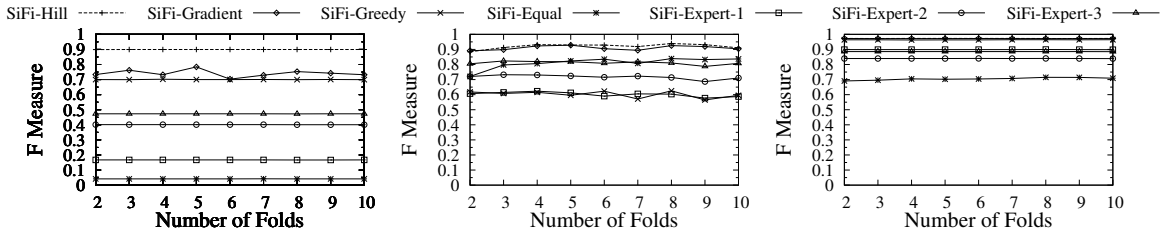
We first compared the effectiveness with existing techniques. Figure 7 reports F-measure values by varying the number of folds. We see that SiFi-Hill can get higher values than OpTrees. For example, in Figure 7(a) the values of SiFi-Hill are around 0.9 while those of OpTrees are around 0.8. This is because OpTrees does not consider the redundancy among similarity functions. In our experiments, there are a lot of optional similarity functions and OpTrees fails to find the optimal similarity functions. We also see that SVM can get the highest values while it consumes the most time for record matching as shown in the following experiment.

Next we compared the efficiency with existing techniques. We used the whole data set as training and testing data. Table 3 reports the results. The training time of SiFi-Hill, OpTrees, SVM respectively refers to the time of computing the optimal eARs, the time of constructing the operator tree, and the time of learning classifier. The testing time respectively refers to the time of executing eRRs, the time of executing the operator tree, and the time of running the classifier. In the training process, we see that SiFi-Hill consumes the least time. For example, on *Cora* SiFi-Hill needs 2018 seconds which is about half of the time of OpTrees and SVM. In the testing process, SiFi-Hill and OpTrees are much more efficient than SVM. For example, on *Restaurant* the elapsed time for SiFi-Hill and OpTrees are less than 8 seconds while SVM needs 221.8 seconds. This is because similarity-join operators can use some filter techniques to efficiently join similar pairs [7]. Thus, SiFi-Hill is more efficient and interpretable for record matching and can also achieve comparable accuracy with machine-learning techniques (SVM).

Table 3: Efficiency Comparison (seconds).

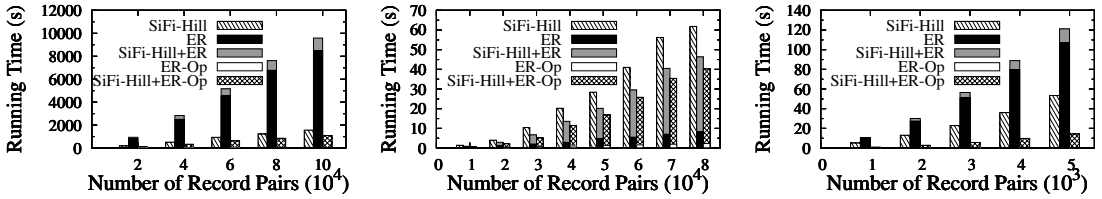
	Cora		*Restaurant*		*DBGen*	
	train	test	train	test	train	test
SiFi-Hill	2018	35	43.2	4.9	17.4	0.85
OpTrees	4555	47	1362	7.6	20.4	0.73
SVM	3610	3035	263.4	221.8	23.6	20.5

[1]In our settings, there are large numbers of candidate functions. When $d > 2$, OpTrees cannot finish in 3 days.



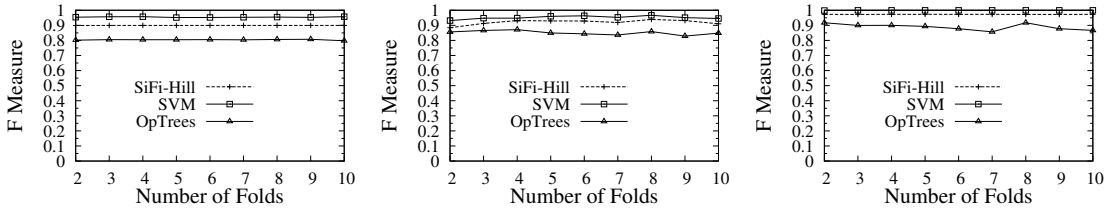
(a) Cora (b) Restaurant (c) DBGen

Figure 5: Comparing F-measure of different methods.



(a) Cora (b) Restaurant (c) DBGen

Figure 6: Comparison for the total running time of different methods.



(a) Cora (b) Restaurant (c) DBGen

Figure 7: Comparison of F-measure with exsiting methods.

7. CONCLUSION

In this paper we have studied the problem of similarity-function identification in record-matching rules for effective entity matching. We proposed to identify the best similarity functions and thresholds to maximize a given objective function. We proposed to detect and eliminate redundancy among similarity functions and thresholds. We also devised efficient algorithms to find the best similarity functions to maximize the eliminated redundancy. The experimental results on both real and synthetic datasets show that our method achieves high performance and result quality.

Acknowledgement. The authors thank the anonymous reviewers for their insightful suggestions. This work was partly supported by the Research Grants Council of the Hong Kong SAR, China, under Grant No. CUHK/419008 and CUHK/419109, the National Natural Science Foundation of China under Grant No. 61003004 and 60873065, the National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302206, National S&T Major Project of China under Grant No. 2011ZX01042-001-002, and the "NExT Research Center" funded by MDA, Singapore, under the research Grant No. WBS:R-252-300-001-490.

8. REFERENCES

[1] http://secondstring.sourceforge.net/.
[2] http://www.dcs.shef.ac.uk/∼sam/simmetrics.html.
[3] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.
[4] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *Proceedings of the 2003 ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 25–27, 2003.
[5] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.
[6] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. Example-driven design of efficient record matching queries. In *VLDB*, pages 327–338, 2007.
[7] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, pages 5–16, 2006.
[8] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWEB*, pages 73–78, 2003.
[9] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD*, pages 475–480, 2002.
[10] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.
[11] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *PVLDB*, 2(1):407–418, 2009.
[12] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.
[13] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.
[14] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.
[15] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Company, 1997.
[16] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.
[17] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, pages 802–803, 2006.
[18] E. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *ICDE*, pages 294–301, 1993.
[19] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.
[20] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.
[21] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD*, pages 350–359, 2002.
[22] W. E. Winkler. Methods for record linkage and bayesian networks. Technical report, Series RRS2002/05, U.S. Bureau of the Census, 2002.

APPENDIX

A. SUMMARY OF NOTATIONS

Table 4: Summary of notations used in this paper.

Notations	Descriptions
R	a relation
a	an attribute in R
r	a record in R
λ	an attribute-matching rule (AR)
λ^i	an implicit attribute-matching rule (iAR)
λ^e	an explicit attribute-matching rule (eAR)
ϕ	a record-matching rule (RR)
ψ	an explicit record-matching rule (eRR)
Φ	a set of record-matching rules (RRs)
Ψ	a set of explicit record-matching rules (eRRs)
f	a similarity function
$\mathcal{F}$	a set of similarity functions
θ	a similarity-function threshold
Θ	a threshold range
E	a set of examples
M	a set of positive examples
D	a set of negative examples
$\overline{M}_\Psi$	a set of record pairs generated by Ψ

B. PROOF

Proof of Theorem 1

PROOF. As $\theta_1 < \theta_2$, we have $\overline{M}_{\lambda_1^e} \supseteq \overline{M}_{\lambda_2^e}$. Thus

$$\overline{M}_{\lambda_1^e} \cap M \supseteq \overline{M}_{\lambda_2^e} \cap M, \ \ \overline{M}_{\lambda_1^e} \cap D \supseteq \overline{M}_{\lambda_2^e} \cap D.$$

As there is no positive example in $\overline{M}_{\lambda_1^e} - \overline{M}_{\lambda_2^e}$, we have

$$\overline{M}_{\lambda_1^e} \cap M = \overline{M}_{\lambda_2^e} \cap M.$$

For any $\psi_1 \in \Psi_1$, if ψ_1 does not contain λ_1^e, Ψ_2 also contains ψ_1; otherwise Ψ_2 replaces it with another eRR ψ_2 by substituting λ_1^e for λ_2^e. As $\overline{M}_\psi = \cap \overline{M}_{\lambda_i}$, we have

$$\overline{M}_{\psi_1} \cap M = \overline{M}_{\psi_2} \cap M, \ \ \overline{M}_{\psi_1} \cap D \supseteq \overline{M}_{\psi_2} \cap D.$$

As $\overline{M}_\Psi = \cup \overline{M}_{\psi_i}$, we have

$$\overline{M}_{\Psi_1} \cap M = \overline{M}_{\Psi_2} \cap M, \ \ \overline{M}_{\Psi_1} \cap D \supseteq \overline{M}_{\Psi_2} \cap D.$$

That is $\mathcal{F}(\Psi_1, M, D) \leq \mathcal{F}(\Psi_2, M, D)$. □

Proof of Corollary 1

PROOF. Consider an eRR $\lambda_1^e(a, f, \theta_1)$ of this iAR. Suppose $\lambda_1^e \notin \mathcal{P}(\lambda^i)$. We prove that λ_1^e can be pruned. Let $V = \{\theta_{max}\} \cup \{f(r[a], r'[a]) | f \in \mathcal{F}, (r, r') \in M\}$. As $\lambda_1^e \notin \mathcal{P}(\lambda^i)$, $\theta_1 \notin V$. As $\theta_1 < \theta_{max}$, we find $\theta_2 \in V$, which is the minimal value in V that is larger than θ_1. Let $\lambda_2^e = (a, f, \theta_2)$. There is no positive examples in $\overline{M}_{\lambda_1^i} - \overline{M}_{\lambda_2^i}$, thus we can prune λ_1^e based on Theorem 1; otherwise, suppose there is a positive example in $\overline{M}_{\lambda_1^i} - \overline{M}_{\lambda_2^i}$ with similarity θ'. Obviously $\theta_1 < \theta' < \theta_2$. Based on the definition of V, $\theta' \in V$, which conflicts that θ_2 is the minimal value in V that is larger than θ_1. □

Proof of Theorem 2

PROOF. Consider a set of RRs with an iRR which contains λ^i. Suppose Ψ_1 is a specificized eRR set, Ψ_2 is another specificized eRR set transformed from Ψ_1 by replacing λ_1^e in Ψ_1 with λ_2^e. Since $\theta_1 > \theta_2$ we have $\overline{M}_{\lambda_1^e} \subseteq \overline{M}_{\lambda_2^e}$, thus

$$\overline{M}_{\lambda_1^e} \cap M \subseteq \overline{M}_{\lambda_2^e} \cap M, \ \ \overline{M}_{\lambda_1^e} \cap D \subseteq \overline{M}_{\lambda_2^e} \cap D.$$

For any record pair $(r, r') \in \overline{M}_{\lambda_2^e} \cap D$, (r, r') should be in $\overline{M}_{\lambda_1^e} \cap D$; otherwise if $(r, r') \in D$, $(r, r') \notin \overline{M}_{\lambda_1^e}$, and as $(r, r') \in \overline{M}_{\lambda_2^e}$, $(r, r') \in \overline{M}_{\lambda_2^e} - \overline{M}_{\lambda_1^e} \subseteq M$, which conflicts with $(r, r') \notin M$ (as $(r, r') \in D$). Therefore, $\overline{M}_{\lambda_1^e} \cap D = \overline{M}_{\lambda_2^e} \cap D$.

With the same idea as the proof in Theorem 1, we have

$$\overline{M}_{\Psi_1} \cap M \subseteq \overline{M}_{\Psi_2} \cap M \ \ \overline{M}_{\Psi_1} \cap D = \overline{M}_{\Psi_2} \cap D.$$

That is $\mathcal{F}(\Psi_1, M, D) \leq \mathcal{F}(\Psi_2, M, D)$. λ_1^e is redundant w.r.t λ_2^e since it cannot get a better objective value than λ_2^e. □

Proof of Theorem 3

PROOF. Consider a set of RRs with an iRR which contains λ^i. Suppose Ψ_1 is a specificized eRR set, Ψ_2 is another specificized eRR set transformed from Ψ_1 by replacing λ_1^e in Ψ_1 with λ_2^e. Since $\theta_1 > \theta_2$ we have $\overline{M}_{\lambda_1^e} \subseteq \overline{M}_{\lambda_2^e}$, thus

$$\overline{M}_{\lambda_1^e} \cap M \subseteq \overline{M}_{\lambda_2^e} \cap M, \ \ \overline{M}_{\lambda_1^e} \cap D \supseteq \overline{M}_{\lambda_2^e} \cap D.$$

With the same idea as the proof in Theorem 1, we have

$$\overline{M}_{\Psi_1} \cap M \subseteq \overline{M}_{\Psi_2} \cap M, \ \ \overline{M}_{\Psi_1} \cap D \supseteq \overline{M}_{\Psi_2} \cap D.$$

That is $\mathcal{F}(\Psi_1, M, D) \leq \mathcal{F}(\Psi_2, M, D)$. λ_1^e is redundant w.r.t λ_2^e since it cannot get a better objective value than λ_2^e. □

Proof of Theorem 4

PROOF. We prove that the SiFi problem is *NP-hard* using a reduction from maximum coverage problem [15]. Given a universal set $\mathcal{U} = \{e_1, e_2, \cdots, e_n\}$ of n elements, a collection $\mathcal{C} = \{S_1, S_2, \cdots, S_m\}$ where $S_i \subseteq \mathcal{U}$ $(i \in [1, m])$, and an integer k, the maximum-coverage problem is to select k sets from $\mathcal{C}$ such that their union has the maximum cardinality. We consider a variant of the maximum-coverage problem that given k collections $\mathcal{C}_1, \cdots, \mathcal{C}_k$ where each collection is the same as $\mathcal{C} = \{S_1, S_2, \cdots, S_m\}$, the goal is to select one set from each collection such that their union has the maximum cardinality. A minor difference from the original problem is that duplicate sets are allowed, e.g. we can select S_1 from both $\mathcal{C}_1$ and $\mathcal{C}_2$. Obviously, the difference can not lead to a larger objective value. Therefore, we can solve the maximum-coverage problem via its variant. Next we reduce the variant to the SiFi problem.

1. Construct a positive example set $M = \{e_1, e_2, \cdots, e_n\}$ and a negative example set $D = \{e_{n+1}, e_{n+2}, \cdots, e_{2n}\}$.
2. The set $\Phi = \{\lambda_1^i, \lambda_2^i, \cdots, \lambda_k^i\}$ contains k RRs where each RR consists of one iAR, and each iAR $\lambda^i : (a, \mathcal{F}, [0, 1])$ has a different attribute from other iARs in Φ and a similarity-function set $\mathcal{F} = \{f_1, f_2, \cdots, f_m\}$.
3. Consider one $\lambda^i : (a, \mathcal{F}, [0, 1])$ in Φ. For each similarity function $f_j \in \mathcal{F}$ $(j \in [1, m])$, we divide the example set $M \cup D$ into two disjoint subsets, S_j and $(M \cup D) - S_j$. Assume that for each example in S_j, the f_j-similarity of the attribute a is θ_1; for each example in $(M \cup D) - S_j$, the f_j-similarity of the attribute a is θ_2 $(\theta_2 < \theta_1)$. Based on the algorithm of computing candidate eAR set in Section 3, we can generate three candidate eARs for λ^i w.r.t f_j, i.e. $\lambda_0^e : (a, f_j, 1)$, $\lambda_1^e : (a, f_j, \theta_1)$ and $\lambda_2^e : (a, f_j, \theta_2)$. The example set that satisfies λ_0^e,λ_1^e,λ_2^e is respectively $\overline{M}_{\lambda_0^e} = \{\}$, $\overline{M}_{\lambda_1^e} = S_j$ and $\overline{M}_{\lambda_2^e} = M \cup D$. We can eliminate λ_0^e since it is threshold-redundant w.r.t λ_1^e. After dealing with all similarity functions in $\mathcal{F}$, we generate the candidate eAR set $\mathcal{P}(\lambda^i) = \{\lambda_j^e : (a, f_j, \theta_1) | j \in [1, m]\} \bigcup \{\lambda_{j+m}^e : (a, f_j, \theta_2) | j \in [1, m]\}$ where $\overline{M}_{\lambda_j^e} = S_j$ and $\overline{M}_{\lambda_{j+m}^e} = M \cup D$ $(j \in [1, m])$.
4. Define the objective function as $\mathcal{F}(\Psi, M, D) = |\overline{M}_\Psi \cap M| - |\overline{M}_\Psi \cap D|$. Any eAR in $\{\lambda_{j+m}^e : (a, f_j, \theta_2) | j \in$

pairs	f_e	f_g	pairs	f_e	f_g	pairs	f_e	f_g
$\text{RP}_{1,2}$	λ_6^e	λ_{12}^e	$\text{RP}_{2,4}$	λ_6^e	-	$\text{RP}_{3,7}$	λ_3^e	λ_8^e
$\text{RP}_{1,3}$	λ_2^e	λ_9^e	$\text{RP}_{2,5}$	λ_4^e	λ_9^e	$\text{RP}_{4,5}$	-	-
$\text{RP}_{1,4}$	λ_6^e	-	$\text{RP}_{2,6}$	λ_6^e	λ_{12}^e	$\text{RP}_{4,6}$	λ_6^e	-
$\text{RP}_{1,5}$	λ_4^e	λ_9^e	$\text{RP}_{2,7}$	λ_6^e	λ_{12}^e	$\text{RP}_{4,7}$	-	-
$\text{RP}_{1,6}$	λ_3^e	λ_{10}^e	$\text{RP}_{3,4}$	λ_6^e	λ_{12}^e	$\text{RP}_{5,6}$	λ_6^e	λ_{12}^e
$\text{RP}_{1,7}$	λ_2^e	λ_8^e	$\text{RP}_{3,5}$	λ_6^e	λ_{10}^e	$\text{RP}_{5,7}$	λ_6^e	λ_{10}^e
$\text{RP}_{2,3}$	λ_6^e	λ_{12}^e	$\text{RP}_{3,6}$	λ_6^e	λ_{12}^e	$\text{RP}_{6,7}$	λ_5^e	λ_{11}^e

Figure 8: A forward index over $\mathcal{P}(\lambda_1^i)$ in Table 2.

$[1, m]\}$ can not lead to the maximum objective value. Otherwise, if λ_{j+m}^e is chosen as an instance of λ^i, then $|\overline{M}_\Psi \cap M| \leq n$ and $|\overline{M}_\Psi \cap D| = n$, thus $F(\Psi, M, D) \leq 0$. But for the case that only eARs in $\{\lambda_j^e : (a, f_j, \theta_1)|j \in [1, m]\}$ are chosen, we have $|\overline{M}_\Psi \cap M| > 0$ and $|\overline{M}_\Psi \cap D| = 0$, thus $F(\Psi, M, D) > 0$. Therefore we can reduce $\mathcal{P}(\lambda^i)$ to $\{\lambda_j^e : (a, f_j, \theta_1)|j \in [1, m]\}$.

The above SiFi problem aims to choose an eAR from each $\mathcal{P}(\lambda^i)$ such that $|\overline{M}_\Psi \cap M| - |\overline{M}_\Psi \cap D|$ is maximum. Since $\mathcal{P}(\lambda^i) = \{\lambda_j^e : (a, f_j, \theta_1)|j \in [1, m]\}$, then for $\lambda_j^e \in \mathcal{P}(\lambda^i)$, we have $\overline{M}_{\lambda_j^e} = S_j \subseteq M$, thus $|\overline{M}_\Psi \cap M| - |\overline{M}_\Psi \cap D| = |(\bigcup_{\lambda^e \in \Psi} \overline{M}_{\lambda^e}) \cap M| - |(\bigcup_{\lambda^e \in \Psi} \overline{M}_{\lambda^e}) \cap D| = \bigcup_{\lambda^e \in \Psi} \overline{M}_{\lambda^e}$. An equivalent statement of the SiFi problem is that given k collections where each one is $\{\overline{M}_{\lambda_j^e}|j \in [1, m]\}$, the goal is to select one set from each collection such that $\bigcup_{\lambda^e \in \Psi} \overline{M}_{\lambda^e}$ is maximum. Since $\overline{M}_{\lambda_j^e} = S_j$ $(j \in [1, m])$, the problem is the same as the variant of the maximum coverage problem. □

C. TIME COMPLEXITY ANALYSIS OF REDUNDANCY DETECTION

We first analyze the time complexity of the algorithm for threshold-redundancy detection. It contains two steps to detect redundant eARs in $\mathcal{P}(\lambda^i)$. The first step is to construct the compressed inverted index CIX and the second step is to use CIX to detect redundancy. For the first step, as there are $|E|$ record pairs and each pair needs $\mathcal{O}(|\mathcal{F}| \cdot log|\mathcal{P}(\lambda^i)|)$ time to insert it into CIX (as it needs to do a binary search to insert λ^i into corresponding position), the total time to construct CIX is $\mathcal{O}(|\mathcal{F}| \cdot log|\mathcal{P}(\lambda^i)| \cdot |E|)$. For the second step, for each $\lambda^e \in \mathcal{P}(\lambda^i)$ the algorithm needs $\mathcal{O}(log|\mathcal{P}(\lambda^i)|)$ to check whether λ^e is redundant, thus the total time of this step is $\mathcal{O}(|\mathcal{P}(\lambda^i)| \cdot log|\mathcal{P}(\lambda^i)|)$. Based on the algorithm of computing candidate eAR set in Section 3, $|\mathcal{P}(\lambda^i)|$ is no larger than $|\mathcal{F}| \cdot |E|$, thus the whole algorithm to detect threshold redundancy needs $\mathcal{O}\big(|\mathcal{F}| \cdot log|\mathcal{P}(\lambda^i)| \cdot |E|\big)$, i.e. $\mathcal{O}\big(|\mathcal{F}| \cdot |K| \cdot |E|\big)$ where $|K| = log(|\mathcal{F}| \cdot |\mathcal{G}_f|)$.

We analyze the time complexity of the algorithm for function redundancy detection. It needs $\mathcal{O}(|\mathcal{F}| \cdot log|\mathcal{P}(\lambda^i)| \cdot |E|)$ time to construct the compressed inverted index and the forward index, and $\mathcal{O}(|\mathcal{F}|^2 \cdot |E|)$ time to compute TL_M and TL_D. When detecting similarity-function redundancy, it enumerates $|\mathcal{P}(\lambda^i)|$ eARs and needs $\mathcal{O}(1)$ to check whether each eAR is redundant w.r.t another group. Since there are $|\mathcal{F}|$ groups, the total time is $\mathcal{O}(|\mathcal{F}| \cdot |\mathcal{P}(\lambda^i)|)$. Since $|\mathcal{P}(\lambda^i)|$ is no larger than $|\mathcal{F}| \cdot |E|$, it needs $\mathcal{O}\big(|\mathcal{F}| \cdot |K| \cdot |E|\big)$ where $|K| = \min\Big(|\mathcal{F}|, log(|\mathcal{P}(\lambda^i)|)\Big)$ to eliminate similarity-function redundancy.

D. PRE-COMPUTING TL_M AND TL_D

To avoid the expensive computation of finding λ_M^e and λ_D^e on the fly, we pre-compute λ_M^e and λ_D^e and store them into two tables TL_M and TL_D. $TL_M(\lambda^e, f)$ stores the maximal eAR λ_M^e in $\mathcal{G}_f$ such that $\overline{M}_{\lambda^e} \cap M \subseteq \overline{M}_{\lambda_M^e} \cap M$. $TL_D(\lambda^e, f)$ stores the maximal eAR λ_D^e in $\mathcal{G}_f$ such that $\overline{M}_{\lambda^e} \cap D \subseteq \overline{M}_{\lambda_D^e} \cap D$. Next we focus on obtaining TL_M and TL_D efficiently.

We first construct a forward index FX over $\mathcal{P}(\lambda^i)$ from E. Each row represents a record pair RP in E and each column represents a similarity function f in F. $FX(\text{RP}, f)$ stores the maximal eAR λ^e in $\mathcal{G}_f$ such that $\text{RP} \vDash \lambda^e$. Figure 8 shows the forward index for the running example. For instance, $FX(\text{RP}_{1,5}, f_g) = \lambda_9^e$ as $\text{RP}_{1,5} \vDash \lambda_9^e$ but $\text{RP}_{1,5} \nvDash \lambda_8^e$. Specially $FX(\text{RP}_{4,5}, f_e) =$ '$-$' means there is no eAR λ^e in $\mathcal{G}_{f_e}$ such that $\text{RP}_{4,5} \vDash \lambda^e$. We analyze the time complexity of constructing FX. *It* has $|E|$ rows and $|\mathcal{F}|$ columns. Consider one cell $FX(\text{RP}, f)$. We need $\mathcal{O}(log|\mathcal{P}(\lambda^i)|)$ time to find the maximal eAR λ^e in $\mathcal{G}_f$ such that $\text{RP} \vDash \lambda^e$. Therefore, the total time is $\mathcal{O}(|\mathcal{F}| \cdot |E| \cdot log|\mathcal{P}(\lambda^i)|)$.

Suppose we want to compute $TL_M(\lambda^e, f)$ $(\lambda^e \in \mathcal{G}_{f_i})$ that is the maximal eAR λ_M^e in $\mathcal{G}_f$ such that $\overline{M}_{\lambda^e} \cap M \subseteq \overline{M}_{\lambda_M^e} \cap M$. Then for each record pair RP in $\overline{M}_{\lambda^e} \cap M$, we have $\text{RP} \vDash \lambda_M^e$. And since $FX(\text{RP}, f)$ is the maximal eAR in $\mathcal{G}_f$ such that $\text{RP} \vDash FX(\text{RP}, f)$, we have $\lambda_M^e \leq FX(\text{RP}, f)$ for each RP. As λ_M^e should be maximal, we have

$$\lambda_M^e = \min_{\text{RP} \in \overline{M}_{\lambda^e} \cap M} FX(\text{RP}, f).$$

Using this equation, we can compute $TL_M(\lambda^e, f)$ for all $\lambda^e \in \mathcal{G}_{f_i}$ incrementally. Let $\lambda^{e(k)}$ be the k-th largest eAR in $\mathcal{G}_{f_i}$. Initially, let $\lambda_M^{e(0)}$ be the maximal eAR in $\mathcal{G}_f$. Suppose we have obtained $\lambda_M^{e(k)}$ for $\lambda^{e(k)}$. Then we compute $\lambda_M^{e(k+1)}$ for $\lambda^{e(k+1)}$ incrementally. As $\overline{M}_{\lambda^{e(k+1)}} \cap M = (\overline{M}_{\lambda^{e(k)}} \cap M) \cup CIX_M(\lambda^{e(k+1)})$, we have

$$\begin{aligned}
\lambda_M^{e(k+1)} &= \min_{\text{RP} \in \overline{M}_{\lambda^{e(k+1)}} \cap M} FX(\text{RP}, f) \\
&= \min\Big(\min_{\text{RP} \in \overline{M}_{\lambda^{e(k)}} \cap M} FX(\text{RP}, f), \min_{\text{RP} \in CIX_M(\lambda^{e(k+1)})} FX(\text{RP}, f)\Big) \\
&= \min\Big(\lambda_M^{e(k)}, \min_{\text{RP} \in CIX_M(\lambda^{e(k+1)})} FX(\text{RP}, f)\Big).
\end{aligned}$$

This equation shows that $\lambda_M^{e(k+1)}$ can be obtained by comparing $\lambda_M^{e(k)}$ and $FX(\text{RP}, f)$ for each $\text{RP} \in CIX_M(\lambda^{e(k+1)})$. Since $|M| = \sum_{\lambda^{e(k)} \in \mathcal{G}_{f_i}} CIX_M(\lambda^{e(k)})$, we only need $\mathcal{O}(|M|)$ time to obtain $TL_M(\lambda^e, f)$ for all $\lambda^e \in \mathcal{G}_{f_i}$. Since the numbers of f and groups are both $|\mathcal{F}|$, we need $\mathcal{O}(|\mathcal{F}|^2 \cdot |M|)$ time to obtain the table TL_M. With the same idea, we can obtain the table TL_D in $\mathcal{O}(|\mathcal{F}|^2 \cdot |D|)$ time. Therefore, the total time of computing TL_M and TL_D is $\mathcal{O}(|\mathcal{F}|^2 \cdot |E|)$.

E. ELIMINATING REDUNDANCY

We find that eliminating threshold redundancy has no effect on similarity-function redundancy. That is if λ_1^e is threshold redundant w.r.t $\mathcal{P}(\lambda^i)$, after eliminating λ_1^e, for any λ_2^e that is similarity-function redundant w.r.t $\mathcal{P}(\lambda^i)$, λ_2^e is still similarity-function redundant w.r.t $\mathcal{P}(\lambda^i) - \{\lambda_1^e\}$. Therefore, we can first eliminate its threshold redundancy, then eliminate its similarity-function redundancy. The correctness is shown in Lemma 1.

LEMMA 1. *Given a* candidate eAR set $\mathcal{P}(\lambda^i)$*, a set M of positive examples and a set D of negative examples, suppose $\lambda_1^e \in \mathcal{P}(\lambda^i)$ is threshold redundant w.r.t $\mathcal{P}(\lambda^i)$ and $\lambda_2^e \in \mathcal{P}(\lambda^i)$ is similarity-function redundant w.r.t $\mathcal{P}(\lambda^i)$. If λ_1^e is eliminated from $\mathcal{P}(\lambda^i)$, then λ_2^e is still similarity-function redundant w.r.t $\mathcal{P}(\lambda^i) - \{\lambda_1^e\}$.*

The eliminating order of function redundancy and threshold redundancy will not change the final results. That is we get the same results for the two cases (1) eliminating function redundancy first and then threshold redundancy; and (2) eliminating threshold redundancy first and then function redundancy. The correctness is shown in Lemma 2

LEMMA 2. *Consider two possible sequences of eliminating redundancy.* Seq_1*: Eliminating threshold redundancy first, then function redundancy;* Seq_2*: Eliminating function redundancy first, then threshold redundancy. Given a* candidate eAR set $\mathcal{P}(\lambda^i)$*, a set M of positive examples and a set D of negative examples, for any $\lambda^e \in \mathcal{P}(\lambda^i)$, we have (1) both of Seq_2 and Seq_2 will eliminate λ^e; (2) neither Seq_1 nor Seq_2 will eliminate λ^e.*

F. PRACTICAL APPLICABILITY

Missing Attribute Values: Our approach can support the case of missing attribute values. This is because given multiple RRs, a record pair is taken as matching if it satisfies one of RRs. A missing attribute value can affect some RRs while others may still work. Consider the RRs in Figure 2. If a value of "`tel`" attribute is missing, ϕ_1 will be affected while ϕ_2 and ϕ_3 still work. Thus if r_6[`tel`] is missing, the matching pair $\text{RP}_{6,7}$ can still be returned based on ϕ_2 (if `email` attributes are same and `addr` attributes are similar).

Combination of Similarity Values: The record matching rules allow a combination of similarity values of a single attribute. This can be achieved by adding a combination of similarity functions into the similarity-function set $\mathcal{F}$. For example, consider $\lambda^i = \{\texttt{name}, \mathcal{F}, [0, 1]\}$. Adding $0.7f_e + 0.3f_j$ into $\mathcal{F}$ makes `name` attribute allow a combination of Edit Similarity and Jaccard Similarity. However, for the record-matching rules with a combination of similarity values of multiple attributes, there is no method that can efficiently find record pairs satisfying such rules [6]. Since we focus on efficient record-matching methods, such record-matching rules are not allowed.

Majority Votes: To make RRs support majority votes, we can define that a record pair satisfies a set Φ of RRs if and only if it satisfies more than a half of the RRs in Φ. As formalized in Theorem 5, Theorems 1, 2, 3 and Corollary 1 also hold for majority votes. Therefore, the algorithms of computing candidate eAR set and eliminating redundancy are applicable for majority votes.

THEOREM 5. *Theorems 1, 2, 3 and Corollary 1 hold for changing* $\overline{M}_\Psi = \bigcup_{\psi\in\Psi} \overline{M}_\psi$ *to* $\overline{M}_\Psi = \bigcup_{\Psi'\subseteq\Psi, |\Psi'|>\frac{|\Psi|}{2}} \left(\bigcap_{\psi\in\Psi'} \overline{M}_\psi\right)$.

PROOF SKETCH. To prove Theorem 1, we only change the line of its proof "As $\overline{M}_\Psi = \cup\overline{M}_{\psi_i}$, we have" to "As $\overline{M}_\Psi = \bigcup_{\Psi'\subseteq\Psi, |\Psi'|>\frac{|\Psi|}{2}} \left(\bigcap_{\psi\in\Psi'} \overline{M}_\psi\right)$, we have". For Theorems 2, 3 and Corollary 1, we do not change their proofs. □

G. TIME COMPLEXITY ANALYSIS OF SiFi-Greedy, SiFi-Gradient AND SiFi-Hill

Let n denote the number of iARs in Φ. All the algorithms need $\mathcal{O}(n \cdot |\mathcal{F}| \cdot |M|)$ time to obtain candidate eAR sets and $\mathcal{O}\big(n \cdot |\mathcal{F}| \cdot |K| \cdot |E|\big)$ time to eliminate redundancy where $|K| = \min\Big(|\mathcal{F}|, log(|\mathcal{P}(\lambda^i)|)\Big)$. Next we analyze the time complexity of finding the best instance. Let $\mathcal{P}^n(\lambda^i)$ denote the candidate eAR set of λ^i without redundancy. For SiFi-Greedy, to compute $\mathcal{Q}(\lambda^i)$, we need enumerate $|\mathcal{P}^n(\lambda^i)|$ eARs and compute the quality of each eAR with $\mathcal{O}(|E|)$ time. At the beginning, there are n iARs in Φ. The number of iARs decreases one at each time. Therefore, the time complexity of SiFi-Greedy is $\mathcal{O}(\sum_{k=1}^{n} k \cdot |\mathcal{P}^n(\lambda^i)| \cdot |E|)$. For SiFi-Gradient, at each iteration we need enumerate $n \cdot 2|\mathcal{F}|$ neighbors and compute the new objective value with $\mathcal{O}(|\Phi| \cdot |E|)$ time. Therefore, the time complexity of SiFi-Gradient is $\mathcal{O}(t \cdot n \cdot |\mathcal{F}| \cdot |\Phi| \cdot |E|)$ where t is the number of iterations. For SiFi-Hill, to check whether the objective value can become larger, we need enumerate $n \cdot |\mathcal{P}^n(\lambda^i)|$ eARs and compute the new objective value with $\mathcal{O}(|\Phi| \cdot |E|)$ times. Therefore, the time complexity of SiFi-Hill is $\mathcal{O}(t' \cdot n \cdot |\mathcal{P}^n(\lambda^i)| \cdot |\Phi| \cdot |E|)$ where t' is the number of iterations. In the worst case, SiFi-Gradient and SiFi-Hill need $|M|$ iterations, but in our experiment, the algorithms can converge with smaller than 10 iterations on both real and synthetic data sets.

H. ADDITIONAL EXPERIMENTS

H.1 Experiment Setup

Datasets: We used three data sets to evaluate our method.

Cora[2] is a collection of citation entries. The data consists of 1875 distinct citations of 191 papers. We selected 9 frequent attributes in our experiment: *author*, *title*, *venue*, *address*, *publisher*, *editor*, *date*, *volume*, *pages*. It had $\frac{1875*1874}{2} = 1,756,875$ record pairs. To eliminate this quadratic cost, we used a similar idea with the canopy method [19]. We concatenated attribute values of each citations to a string and used jaccard similarity based on tokens to quantify the similarity of two strings. We eliminated record pairs whose similarity are either equal to 1 or no larger than 0.1. After this process, the total number of record pairs was 184,738 with 14,358 positive pairs.

Restaurant[3] is a collection of restaurant records. The data contains 864 distinct records of 752 restaurants. Each record has five attributes: *name*, *addr* (restaurant address), *phone*, *city*, and *type*. It had $\frac{864*863}{2} = 372,816$ record pairs. We used the same method as *Cora* to reduce the number of record pairs to 87,492 with 106 positive pairs.

DBGen [3] is a random mailing-list generator. The generated mailing-list has 10 attributes, *ssn* (social security number), *fname* (first name), *minit* (middle initial), *lname* (last name), *stnum* (street number), *stadd* (street name), *apmt* (apartment number), *city*, *state* and *zip*. We generate data set by setting *Number of Records* to 1000 and *Number of Clusters* to 100, and keeping other parameters by default. Using the same method as *Cora*, we construct a data set of 5,497 record pairs with 3,071 positive pairs.

Similarity Metrics: A large number of similarity functions are proposed to quantity string similarity. *SecondString* [1] and *SimMetrics* [2] are two open-source Java packages that implement a large collection of string functions. In particular, *SimMetrics* provides a consistent interface layer that returns a normalized similarity measure from 0 to 1, 0 being entirely different, 1 being identical. We used *SimMetrics* package in our experiment. For token-based similarity functions, we need to consider different ways of tokenization, such as tokenizing by space, q-gram, etc. Totally we selected 26 similarity functions. In the case of null values, we

[2] http://www.cs.umass.edu/∼mccallum/data/cora-refs.tar.gz

[3] http://www.cs.utexas.edu/users/ml/riddle/data

RR set for the *Cora* data set

$1 : author^i \wedge title^i \wedge venue^e$
$2 : author^i \wedge title^e \wedge venue^i$
$3 : author^e \wedge title^i \wedge venue^i$
$4 : author^i \wedge title^i \wedge venue^i \wedge date^e$
$5 : author^i \wedge title^i \wedge venue^i \wedge volumn^e \wedge pages^e$
$6 : author^i \wedge title^i \wedge publisher^e \wedge editor^e \wedge date^e$
$7 : author^i \wedge title^e \wedge publisher^e \wedge editor^e \wedge volume^e \wedge pages^e$

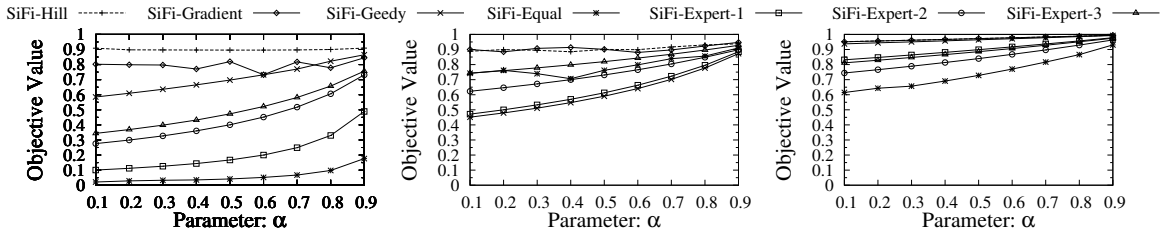
RR set for the *Restaurant* data set

$1 : name^e \wedge addr^i$
$2 : name^i \wedge addr^e$
$3 : name^i \wedge addr^i \wedge city^e$
$4 : name^i \wedge addr^i \wedge city^i \wedge type^e$

RR set for the *DBGen* data set

$1 : fname^i \wedge lname^e \wedge zip^e$
$2 : fname^e \wedge lname^i \wedge zip^e$
$3 : fname^e \wedge lname^e \wedge zip^i$
$4 : fname^e \wedge minit^e \wedge lname^e$
$5 : fname^i \wedge stnum^e \wedge stadd^i \wedge city^e$
$6 : fname^i \wedge stnum^i \wedge stadd^e \wedge city^e$
$7 : lname^i \wedge stnum^e \wedge stadd^i \wedge city^e$
$8 : lname^i \wedge stnum^i \wedge stadd^e \wedge city^e$
$9 : fname^i \wedge stnum^e \wedge stadd^i \wedge city^i \wedge state^e$
$10 : fname^i \wedge stnum^i \wedge stadd^e \wedge city^i \wedge state^e$
$11 : lname^i \wedge stnum^e \wedge stadd^i \wedge city^i \wedge state^e$
$12 : lname^i \wedge stnum^i \wedge stadd^e \wedge city^i \wedge state^e$

Figure 9: RR sets for the *Cora*, *Restaurant*, *DBGen* data sets.



(a) Cora (b) Restaurant (c) DBGen

Figure 10: Comparison with baseline methods for different objective functions.

suppose the similarity between two null values is 1 and the similarity between a null value and a non-null value is 0.

Record-matching Rule Set: To reduce user effort, all iARs are in the form of $(a, \mathcal{F}, [0, 1])$ where $\mathcal{F}$ consists of 26 similarity functions. Figure 9 shows three RR sets for *Cora*, *Restaurant*, *DBGen* respectively. To save space, we denote $(a, f_=, 1)$ as a^e and $(a, \mathcal{F}, [0, 1])$ as a^i. The RR set for *Cora* consists of seven RRs with three iARs, $author^i$, $title^i$ and $venue^i$. The RR set for *Restaurant* consists of four RRs with three iARs, $name^i$, $addr^i$ and $city^i$. The RR set for *DBGen* consists of twelve RRs with six iARs, $fname^i$, $lname^i$, $stnum^i$, $stadd^i$, $city^i$ and zip^i. Note that we ignore the attribute *phone* and the attribute *ssn* in the RR set of *Restaurant* and *DBGen* respectively since we found that simply returning record pairs that share the same *phone* (*ssn*) can lead to an acceptable result. The RR sets of *Restaurant* and *DBGen* are formulated by experts based on domain knowledge. The RR set of *Cora* is deduced from an initial set of matching dependencies [11]. A matching dependency over one relation, $\lambda_1 \wedge \lambda_2 \cdots \wedge \lambda_n \rightarrow a_1, a_2, \cdots, a_m$, denotes if two records satisfy $\lambda_1 \wedge \lambda_2 \cdots \wedge \lambda_n$, then it identifies a set of attributes $a_1, a_2, \cdots, a_m$ where λ_i is either iAR or eAR. For example, $\underline{publisher^e \wedge editor^e \wedge date^e} \rightarrow venue$ denotes if two records have the same values on *publisher*, *editor* and *date* attributes, then they should have the same values on *venue*. If we have another matching dependency $author^i \wedge title^i \wedge \underline{publisher^e \wedge editor^e \wedge date^e} \rightarrow A$, where A contains all the attributes of the relation, then we can deduce the record-matching rule $author^i \wedge title^i \wedge venue^e \rightarrow A$. When the number of deduced rules is large, the algorithm can select the top-k rules based on some heuristic metrics such as the diversity of rules.

User Study: We chose three students, called Expert 1, Expert 2, Expert 3 respectively, from our research group to do user study. They are quite familiar with similarity metrics and data sets. We asked them to formulate the eARs for the iARs in Figure 9. Table 5 shows the result. Note that there are only six candidate functions. Experts tend to select edit-based similarity functions such as Edit Similarity and Soundex for the iAR whose attribute value consists of a small number of tokens (e.g. Expert 1 selects Edit Similarity f_e for $author^i$) and select token-based similarity functions such as Jaccard Similarity and Cosine Similarity for the iAR whose attribute values consists of a large number of tokens (Expert 2 selects Jaccard Similarity f_j for $title^i$). We see eARs differ a lot in both similarity functions and thresholds.

Table 5: The eARs formulated by three experts for iARs in Figure 9 (f_e: Edit Similarity, f_j: Jaccard Similarity, f_g: Gram-based Similarity, f_c: Cosine Similarity [8], f_w: Jaro Winkler [8], f_s: Soundex [17]).

	Expert 1	Expert 2	Expert 3
Cora	$(author, f_e, 0.8)$ $(title, f_j, 0.8)$ $(venue, f_e, 0.85)$	$(author, f_w, 0.75)$ $(title, f_j, 0.7)$ $(venue, f_g, 0.75)$	$(author, f_w, 0.8)$ $(title, f_c, 0.6)$ $(venue, f_e, 0.7)$
Restaurant	$(name, f_e, 0.8)$ $(addr, f_j, 0.8)$ $(city, f_e, 0.8)$	$(name, f_w, 0.75)$ $(addr, f_j, 0.7)$ $(city, f_g, 0.8)$	$(name, f_w, 0.8)$ $(addr, f_c, 0.7)$ $(city, f_e, 0.8)$
DBGen	$(fname, f_s, 0.8)$ $(lname, f_s, 0.8)$ $(stnum, f_e, 0.9)$ $(stadd, f_j, 0.8)$ $(city, f_e, 0.8)$ $(zip, f_e, 0.8)$	$(fname, f_e, 0.8)$ $(lname, f_e, 0.8)$ $(stnum, f_e, 0.9)$ $(stadd, f_j, 0.7)$ $(city, f_g, 0.75)$ $(zip, f_e, 0.8)$	$(fname, f_e, 0.7)$ $(lname, f_e, 0.7)$ $(stnum, f_e, 0.9)$ $(stadd, f_c, 0.7)$ $(city, f_e, 0.8)$ $(zip, f_e, 0.8)$

All the algorithms were implemented in C++ and compiled using GCC 4.2.3 with -O3 flag. All the experiments were run on a Ubuntu machine with an Intel Core 2 Quad X5450 3.00GHz processor and 4 GB memory.

H.2 Evaluation of Objective Functions

We evaluated the effectiveness of our methods with different objective functions. We used a family of objective functions, i.e. $\frac{1}{\alpha * \frac{1}{p} + (1-\alpha)\frac{1}{r}} (\alpha \in (0, 1))$ where α is a parameter to tune the importance of precision and recall. In the case that we require the returned record pairs have higher precision, we can specify a larger α; on the contrary, if we require the returned record pairs miss fewer matching record pairs, we can specify a smaller α. Figure 10 shows the 5-cross validation results on three data sets. We can see SiFi-Hill is always superior to other methods, and has more stable values with changes of objective functions.