Extending Matching Rules with Conditions

Shaoxu Song
The Hong Kong University of Science and Technology
sshaoxu@cse.ust.hk

Lei Chen
The Hong Kong University of Science and Technology
leichen@cse.ust.hk

Jeffrey Xu Yu
The Chinese University of Hong Kong
yu@se.cuhk.edu.hk

ABSTRACT

Matching dependencies (MDs) have recently been proposed [10] in order to make dependencies tolerant to various information representations, and proved [13] useful in data quality applications such as record matching. Instead of strict identification function in traditional dependency syntax (e.g., functional dependencies), MDs specify dependencies based on similarity matching quality. However, in practice, MDs may still be too strict and only hold in a subset of tuples in a relation. Thereby, we study conditioning MDs in a subset of tuples, called *conditional matching dependencies* (CMDs), which bind matching dependencies only in a certain part of a table. Compared to MDs, CMDs have more expressive power that enable them satisfy wider application needs.

In this paper, we study several important theoretical and practical issues of CMDs, including inferring CMDs, the irreducible CMDs with less redundancy, the discovery of CMDs from data, and so on. Through an extensive experimental evaluation in real data sets, we demonstrate the efficiency of proposed CMDs discovery algorithms.

1. INTRODUCTION

Data dependencies, traditionally used for schema design and integrity constraints, are recently revisited for improving data quality [3, 8, 10]. However, traditional dependencies based on identification function, such as functional dependencies (FDs), often fail in such applications, due to various information representations and formats, especially in the Web data. For example, Joy Alice and J. Alice may denote the same person in the real world. Therefore, matching dependencies (MDs) [10] have recently been proposed for data quality applications, such as record matching. In order to be tolerant to various information formats, MDs target on dependencies with respect to similarity matching quality, instead of identification functions in traditional dependency syntax. For example, we consider a relation schema Contacts(name, address, institute, ssn), where tuples with the same ssn denote the same person in the real world. A typical MD over Contacts can be

$$\text{MD} \quad [\mathsf{name} \approx] \wedge [\mathsf{institute} \approx] \rightarrow [\mathsf{ssn} \rightleftharpoons].$$

It states that if any two tuples from Contacts have *similar* (i.e., $\approx$) name and institute, then they must denote the same person in real world with *identical*[1] (i.e., $\rightleftharpoons$) ssn. The MDs, which cooperate similarity operator $\approx$ and matching operator $\rightleftharpoons$, have been proved useful in real applications, such as record matching [13].

Note that the original MDs specify dependency constraints on all the tuples of a relation. In practice, however, it might be too strict, that is, a MD is often valid in a subset of tuples. For example, in Table 1, the above MD on name and institute holds in the tuples t_1 to t_5 whose institute is similar to UST, while t_7 and t_8 do not satisfy this MD since they share similar name and institute but not identical ssn. It motivates us to study the MDs specified in a subset of tuples instead of the entire relation. In fact, conditioning dependencies in a subset of tuples has been highlighted for data quality applications, such as functional dependencies with conditions [3], inclusion dependencies with conditions [5], and sequential dependencies with conditions [16]. The basic idea of these condition extensions is making the dependencies, e.g., FDs that originally hold for the whole table, valid only for a subset of tuples.

In this study, we propose conditional matching dependencies (CMDs), which declare matching dependencies on a subset of tuples specified by conditions.

Examples

A CMD with conditions on determinant can be

$$\text{CMD} \quad [\mathsf{name} \approx *] \wedge [\mathsf{institute} \approx \mathsf{UST}] \rightarrow [\mathsf{ssn} \rightleftharpoons *],$$

where $*$ is a virtual value that is similar/identical to any value. Intuitively, a CMD requires that two tuples should not only be similar on certain attributes, i.e., $\approx$, but also be similar to the specified conditions, e.g., UST. It states that only in the institute of UST, if two tuples have similar name (and similar to $*$), then they must denote the same contact with identical ssn (also identical to $*$).

Moreover, according to the data set in Table 1, we may find another CMD with respect to the condition of Alice on

[1]could also be identified via update with dynamic semantics [10, 13]. Without loss of generality, "identical" in the following also interpret the dynamic semantics via update, while "not identical" means not identified even via update of dynamic semantics.

Table 1: Example instance of Contacts

tid	name	address	institute	ssn
t_1	J.C. Alice	#7, C.R.	UST, HK	111
t_2	J. Alice	No. 7, Central Road	UST	111
t_3	Joy Alice	No. 7, Central Rd.	UST	111
t_4	Alice	7th, Central Road	UST	111
t_5	Miss Alice	7th, Central Rd.	UST	111
t_6	Prof. Alice	8th, Clear Water Bay	CityU	222
t_7	Prof. Alex	No. 7, Central Road	PloyU	333
t_8	Bob Alex	No. 7, Central Road	PloyU	444

attribute name,

$$\text{CMD}_1 \quad [\mathsf{name} \approx \mathsf{Alice}] \wedge [\mathsf{address} \approx *] \rightarrow [\mathsf{ssn} \rightleftharpoons *].$$

For any two tuples, whose name values are similar with each other and also similar to Alice, i.e., t_1 to t_6 in Table 1, if their address values are also similar, then they must be identical on ssn. Such dependencies from name, address to ssn might not be valid over other subsets of tuples in Table 1, e.g., t_7, t_8 with condition Alex.

A CMD with conditions on dependent can be

$$\text{CMD}_2 \quad [\mathsf{name} \approx \mathsf{Alice}] \wedge [\mathsf{address} \approx \mathsf{7th, Central\ Rd.}] \rightarrow [\mathsf{ssn} \rightleftharpoons \mathsf{111}]$$

It states that for any two tuples agreeing the left-hand-side of CMD_2, their ssn values must be identical with each other and also should be identical to 111.

Applications

In contrast to FDs used for schema design, MDs are proposed for various data quality applications. Similar to MDs, the CMDs can also be applied in these data quality applications, with more expressive power.

Record linkage. Dependencies can be used as matching rules in a rule-based method of record linkage [18, 29, 6]. We can directly use MDs (as well as CMDs) as matching rules [13]. It tells how the attributes in a relation should be compared in order to identify duplicate records. As mentioned, CMDs can address more dependencies which only hold in a subset of tuples, that is, more expressive power with respect to matching rules. For example, by using the above MD, both (t_2, t_3) and (t_7, t_8) in Table 1 are identified as pairs of duplicates (should having same ssn), while our CMD_1 can successfully tell that (t_2, t_3) are duplicates but (t_7, t_8) are not.

Violation detection. As integrity constraints, it is natural to use CMDs for violation detection. Tuples that do not follow CMDs are detected as violations, that is, those ones agree the left-hand-side of CMDs but disagree the right-hand-side. Compared with MDs, the proposed CMDs have more expressive power with respect to integrity constraints. It requires that the right-hand-side attributes should not only be identical but also identical to certain values, e.g., $[\mathsf{ssn} \rightleftharpoons 111]$ in the above CMD_2. Those tuples agreeing left-hand-side of CMD_2 and having same ssn but not equal to 111 are then detected as violations, which are ignored by MDs with $[\mathsf{ssn} \rightleftharpoons *]$.

Observations

In this paper, we propose a novel class of dependencies considering both similarity matching quality and condition. To motivate our work, we study the following observations.

The first observation is the relationship among conditions. To formally define *conditional matching dependencies* (CMDs), we introduce syntax with conditions, e.g., $[\mathsf{name} \approx \mathsf{Alice}]$. Given a finite domain of an attribute, two conditions may cover similar semantics of constraints. For example, any value similar to Joy Alice is always similar to Alice (see Example 2.1 for details). We say Alice dominates Joy Alice with respect to the domain of name attribute.

The second observation is the relationship of CMDs. For example, according to the dominating relationship between Alice and Joy Alice, we may imply the CMD with condition of Joy Alice based on another CMD with condition Alice (see Example 3.1 in Section 3 for details). In other words, given a set of CMDs, one may imply another, i.e., redundancy. Obviously, a concise set of CMDs with less redundancy is always preferred by real applications in order to reduce unnecessary overhead. It raises the problem of finding irreducible CMDs. Moreover, in practice, it is rather difficult to generate CMDs by users' domain knowledge, especially getting those specific values as conditions. Similar to discovering FDs [22, 19, 30], a practical way is to give a sample data which embeds possible CMDs. The discovery problem is then to find those CMDs from the sample data. Again, it is promising to find irreducible CMDs from the sample data.

Contributions

With the above observations, it is notable that conditional matching dependencies (CMDs) are not simple extensions to conditional functional dependencies (CFDs) [3]. The proposed CMDs have to handle dominating relationships between domain values, in both inferring and discovering CMDs. Moreover, similarity metrics in CMDs make the computation of agree and error rates difficult, since grouping of tuples by equal values for CFDs is no longer valid for CMDs. Our main contributions are summarized as follow.

- We propose conditional matching dependencies (CMDs). A comprehensive syntax system is introduced including formal definitions of CMDs and dominating relationships of conditions.
- We introduce the logical implication of CMDs. Inference rules including augmentation and left-dominating are presented, which can be used in discovery problem. The concept of irreducible CMDs, which are irreducible and imply others, is also proposed in order to reduce redundancy.
- We investigate the discovery of CMDs from a sample data. Unfortunately, a set of all irreducible CMDs discovered from data can be exponentially large in size. Thereby, we develop pruning algorithms based on the implication properties, in order to improve the discovery performance.
- We report an extensive experimental evaluation in real data sets. Our advanced discovery algorithms can achieve several orders of magnitude improvement compared with the straight-forward method.

The remainder of this paper is organized as follows. First, in Section 2, we introduce syntax and formal definitions of

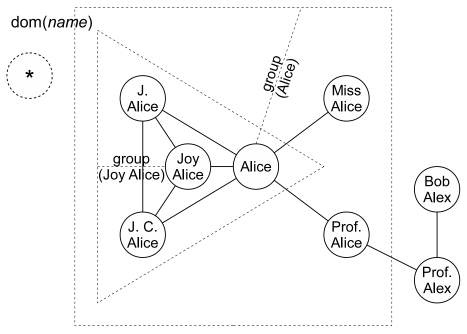


Figure 1: Similarity graph of dom(name)

conditional matching dependencies. Section 3 presents the logical implication of CMDs and studies the discovery of irreducible CMDs. An extensive experimental evaluation is then reported in Section 4. Finally, we discuss the related work in Section 5 and conclude this paper in Section 6.

2. SYNTAX AND SEMANTICS

In this section, we introduce the notations and definitions for conditional matching dependencies. We consider a relation schema R, where each attribute A has a finite domain denoted by $\mathsf{dom}(A)$.

A *similarity operator* $\approx$ on an attribute A is defined on the domain of A,

$$\approx: \mathsf{dom}(A) \times \mathsf{dom}(A) \rightarrow \{\mathsf{true}, \mathsf{false}\},$$

which satisfies *reflexivity*, i.e., $a \approx a$, and *symmetry*, i.e., if $a \approx b$ then $b \approx a$, where $a, b \in \mathsf{dom}(A)$. It indicates true if two values are *similar*. This operator can be domain-specific or any similarity metrics, such as edit distance and cosine similarity, with a predefined threshold. Since it is not the focus of this paper to find perfect similarity operator for a given attribute, in the following, we assume a similarity operator for each attribute. For a set X of attributes, the similarity operator $\approx$ indicates true, *iff* the similarity operators on all $A \in X$ indicate true.

A *matching operator* $\rightleftharpoons$ on an attribute A is also defined on the domain of A. It indicates true if two values are *identical*. It is notable that they could also be identified via update with dynamic semantics [10, 13].

A *matching dependency* (MD) has a form

$$[X \approx] \rightarrow [Y \rightleftharpoons],$$

where $X \subseteq R, Y \subseteq R$, and $\approx, \rightleftharpoons$ denotes the corresponding similarity/matching operators on attributes of X and Y, respectively. It states that for any two tuples from an instance of relation R, if they are similar on attributes X, then their Y values should be identical.

We say a MD $[X \approx] \rightarrow [Y \rightleftharpoons]$ *holds* in an instance I of relation R, or I *satisfies* a MD $[X \approx] \rightarrow [Y \rightleftharpoons]$, denoted by $I \vDash [X \approx] \rightarrow [Y \rightleftharpoons]$, if $\forall t_1, t_2 \in I$, $t_1[X] \approx t_2[X]$ implies $t_1[Y] \rightleftharpoons t_2[Y]$.

Conditional Matching Dependencies

We now extend the above matching dependencies with conditions, that is, MDs conditionally hold in a subset of tuples instead of the entire table. To introduce the formal definition, we first define a virtual value $*$ in $\mathsf{dom}(A)$ for each attribute A, which is similar/identical to all the values in $\mathsf{dom}(A)$, i.e., $* \approx a, * \rightleftharpoons a, \forall a \in \mathsf{dom}(A)$.

DEFINITION 2.1. *A conditional matching dependency* (CMD) *has a form*

$$[X \approx x] \rightarrow [Y \rightleftharpoons y],$$

where x and y are domain values of $X \subseteq R$ and $Y \subseteq R$, having $x \in \mathsf{dom}(X)$ and $y \in \mathsf{dom}(Y)$, respectively.

It states that for any two tuples from an instance I of relation R, if they are similar on attributes X and similar to x as well, then their Y values should be identical with each other and also identical to y. We have presented several examples of CMDs in the introduction. For instance, CMD_1 does not specify constraints on all the tuples in a table, but only those tuples whose name values are similar to Alice. As special cases of CMDs, all the MDs can be represented by CMDs syntax, i.e., $[X \approx *] \rightarrow [Y \rightleftharpoons *]$, since virtual value $*$ is always similar/identical to any value.

For any $t_1, t_2 \in I$, if $t_1[X] \approx t_2[X]$, $t_1[X] \approx x$ and $t_2[X] \approx x$, we say that t_1, t_2 *agree* the condition $[X \approx x]$, denoted by $(t_1, t_2) \asymp [X \approx x]$; otherwise, not agree, denoted by $(t_1, t_2) \not\asymp [X \approx x]$. If $t_1[Y] \rightleftharpoons t_2[Y] \rightleftharpoons y$ as well, we say that t_1, t_2 *agree* the CMD $[X \approx x] \rightarrow [Y \rightleftharpoons y]$, denoted by $(t_1, t_2) \asymp [X \approx x] \wedge [Y \rightleftharpoons y]$.

Formally, an instance I of relation R *satisfies* a CMD, denoted by $I \vDash [X \approx x] \rightarrow [Y \rightleftharpoons y]$, if $\forall t_1, t_2 \in I$, $(t_1, t_2) \asymp [X \approx x]$ implies $(t_1, t_2) \asymp [Y \rightleftharpoons y]$. That is, $\forall t_1, t_2 \in I$, if $t_1[X] \approx t_2[X]$, $t_1[X] \approx x$ and $t_2[X] \approx x$, then $t_1[Y] \rightleftharpoons t_2[Y] \rightleftharpoons y$. Note that any two tuples $(t_1, t_2) \not\asymp [X \approx x]$ always satisfy a dependency $(t_1, t_2) \vDash [X \approx x] \rightarrow [Y \rightleftharpoons y]$ but do not agree the dependency $(t_1, t_2) \not\asymp [X \approx x] \wedge [Y \rightleftharpoons y]$.

Dominating Relationships

According to the above definition, any value $x \in \mathsf{dom}(X)$ can be specified as a condition for CMDs. However, as illustrated in the following example, two arbitrary values of X as conditions in CMDs may overlap and introduce redundancy with respect to dependency semantics.

EXAMPLE 2.1. *We consider all the values in a finite domain of attribute* name. *As presented in Figure 1, each node denotes a value of* name, *and the edge between two nodes means that they are similar, i.e., $\approx$. According to the similarity graph, any value similar to* $\mathsf{Joy\ Alice}$ *always be similar to* Alice *as well. Consequently, the subset of tuples specified by using condition* $\mathsf{Joy\ Alice}$ *should be a subset of the tuples specified by* Alice, *i.e., redundancy on dependency semantics, which will be discussed in detail in the following Example 3.1.*

Motivated by the above example of redundancy issues, we introduce an *order relation* of conditions, namely *dominating relationships*. First, we formalize the set of all the domain values that agree a given condition x. A *group* of a value x on attributes X, denoted by $\mathsf{group}(x)$, is a set of values with respect to $\approx$ in $\mathsf{dom}(X)$ such that for any $x_1, x_2 \in \mathsf{group}(x)$, if $x_1 \approx x_2$, then $x_1 \approx x$ and $x_2 \approx x$. That is, if $x_1 \approx x_2$, then $(x_1, x_2) \asymp [X \approx x]$.

CLAIM 2.1. *For any $x_1 \in \mathsf{dom}(X)$, we have $x_1 \in \mathsf{group}(x)$, iff $x_1 \approx x$.*

The group of virtual value $*$ is $\mathsf{group}(*) = \mathsf{dom}(X)$, which includes all the values. An order relation $\succeq$ is then defined on values based on their corresponding groups.

DEFINITION 2.2. *Let x_3 and x_4 be two values in $\mathsf{dom}(X)$. If $\mathsf{group}(x_3) \supseteq \mathsf{group}(x_4)$ with respect to $\approx$, we say x_3 dominates x_4, denoted by $x_3 \succeq x_4$.*

It states that any two domain values agreeing condition x_4, e.g., $(x_1, x_2) \asymp [X \approx x_4]$ must also agree the condition x_3, i.e., $(x_1, x_2) \asymp [X \approx x_3]$. For example, as illustrated in Figure 1, the group of Joy Alice is a subset of Alice group, i.e., Alice $\succeq$ Joy Alice, which has the following semantics. Any value similar to Joy Alice must be similar to Alice, according to Claim 2.1. Any two values in group(Joy Alice) that are similar, e.g., (J.Alice, J.C.Alice) $\asymp$ [name $\approx$ Joy Alice], should belong to group(Alice) as well, i.e., (J.Alice, J.C.Alice) $\asymp$ [name $\approx$ Alice].

Recall that the similarity operator $\approx$ on X indicates true iff each $A \in X$ is similar, i.e., $x_1 \approx x$ iff $x_1[A] \approx x[A], \forall A \in X$. In other words, according to Claim 2.1, any value $x_1 \in \mathsf{group}(x)$ always has $x_1[A] \in \mathsf{group}(x[A]), \forall A \in X$. Consequently, it is natural to have the following claim.

CLAIM 2.2. *We have $x_3 \succeq x_4$, iff $\forall A \in X, x_3[A] \succeq x_4[A]$.*

Since the virtual value $*$ is similar to all the values of X, the group of $*$ is a superset of any other.

CLAIM 2.3. *For any $x \in \mathsf{dom}(X)$, we have $* \succeq x$.*

3. IRREDUCIBLE CMDs

In this section, we discuss logical implication for inferring CMDs, which raises the problem of finding irreducible CMDs. Before introducing technique details, we first illustrate the following motivation example. As discussed, due to the dominating relationships of conditions, there may exist redundant dependency semantics among CMDs.

EXAMPLE 3.1. *Consider the following CMDs,*

$$\text{CMD}_1 \quad [\mathsf{name} \approx \mathsf{Alice}] \wedge [\mathsf{address} \approx *] \to [\mathsf{ssn} \rightleftharpoons *],$$

$$\text{CMD}_2 \quad [\mathsf{name} \approx \mathsf{Joy\ Alice}] \wedge [\mathsf{address} \approx *] \to [\mathsf{ssn} \rightleftharpoons *],$$

$$\text{CMD}_3 \quad [\mathsf{name} \approx \mathsf{Alice}] \wedge [\mathsf{address} \approx *] \wedge [\mathsf{institute} \approx *] \to [\mathsf{ssn} \rightleftharpoons *].$$

For any two tuples with similar address, according to CMD$_1$, if their name values are similar with each other and similar to Alice as well, then they should also agree [ssn $\rightleftharpoons$ $*$]. Similar dependency semantics are embedded in CMD$_2$ and CMD$_3$.

Suppose that we have Alice $\succeq$ Joy Alice with respect to the finite domain of name, i.e., any two tuples $(t_1, t_2) \asymp$ [name $\approx$ Joy Alice] must agree $(t_1, t_2) \asymp$ [name $\approx$ Alice] as well. According to CMD$_1$, if two tuples agree $(t_1, t_2) \asymp$ [name $\approx$ Alice] and $(t_1, t_2) \asymp$ [address $\approx *$], they must also agree $(t_1, t_2) \asymp$ [ssn $\rightleftharpoons *$]. In other words, CMD$_2$ can be inferred by CMD$_1$. In fact, as we illustrate after Lemma 3.2, any two tuples satisfying $(t_1, t_2) \vDash$ CMD$_1$ must satisfy $(t_1, t_2) \vDash$ CMD$_2$. Thereby, CMD$_2$ is redundant with respect to CMD$_1$.

Similarly, any two tuples agreeing the left-hand-side of CMD$_3$ with [institute $\approx *$] must also agree the left-hand-side of CMD$_1$ which does not require institute to be similar. Consequently, CMD$_3$ can be inferred according to CMD$_1$ as well.

In the following, we give a formal mechanism for inferring CMDs, and introduce the irreducible CMDs with less redundancy. The discovery of these irreducible CMDs, given a sample relation instance, is then developed.

3.1 Rationale

As presented in Example 3.1, any two tuples satisfying CMD$_1$ must satisfy CMD$_2$. We say that CMD$_1$ can logically imply CMD$_2$.

Inferring CMDs

Let Σ_1 and Σ_2 be two sets of CMDs over relation R. We say Σ_1 *implies* Σ_2, denoted by $\Sigma_1 \vDash \Sigma_2$, if for all instances I over relation R, $I \vDash \Sigma_1$ implies $I \vDash \Sigma_2$. According to the properties of CMDs, we can investigate the following inference rules.

LEMMA 3.1 (AUGMENTATION). *For all instances I over relation R, if $I \vDash [X \approx x] \to [Y \rightleftharpoons y]$, then $I \vDash [X \approx x] \wedge [Z \approx z] \to [Y \rightleftharpoons y]$.*

PROOF. (sketch) Since any tuple pairs agreeing $[X \approx x] \wedge [Z \approx z]$ must also agree $[X \approx x]$, the tuples in I agreeing on $[X \approx x] \wedge [Z \approx z]$ should be a subset of those agreeing $[X \approx x]$. For those tuples satisfying $[X \approx x] \to [Y \rightleftharpoons y]$, any subset of them can also determine $[Y \rightleftharpoons y]$. □

For instance, CMD$_1$ in Example 3.1 can imply CMD$_3$, denoted by CMD$_1$ $\vDash$ CMD$_3$. That is, for any t_1 and t_2 from instance of relation R, if (t_1, t_2) satisfy CMD$_1$, it always has $(t_1, t_2) \vDash$ CMD$_3$ as well. To illustrate this conclusion, we consider three possible cases:

- if $(t_1, t_2) \not\asymp$ [name $\approx$ Alice] $\wedge$ [address $\approx *$], then (t_1, t_2) satisfy both CMD$_1$ and CMD$_3$, since they do not agree the left-hand-side;
- if $(t_1, t_2) \asymp$ [name $\approx$ Alice] $\wedge$ [address $\approx *$] but $(t_1, t_2) \not\asymp$ [institute $\approx *$], then (t_1, t_2) satisfy CMD$_3$ too;
- if $(t_1, t_2) \asymp$ [name $\approx$ Alice] $\wedge$ [address $\approx *$]$\wedge$[institute $\approx *$], then we have $(t_1, t_2) \asymp$ [ssn $\rightleftharpoons *$] according to CMD$_1$, i.e., (t_1, t_2) satisfy CMD$_3$ again.

Consequently, we have $(t_1, t_2) \vDash$ CMD$_3$ in all three possible cases.

LEMMA 3.2 (LEFT-DOMINATING). *For all instances I over relation R, if $I \vDash [X \approx x_1] \to [Y \rightleftharpoons y]$, then, for any x_2 such that $x_1 \succeq x_2$, we have $I \vDash [X \approx x_2] \to [Y \rightleftharpoons y]$.*

PROOF. (sketch) According to the definition of dominating, any tuple pair agreeing $[X \approx x_2]$ must agree $[X \approx x_1]$ as well. If $[X \approx x_1]$ is agreed, then $[Y \rightleftharpoons y]$ should also be concluded for the tuple pair according to $[X \approx x_1] \to [Y \rightleftharpoons y]$. □

For example, we have CMD$_1$ $\vDash$ CMD$_2$ in Example 3.1. That is, for any tuple pair satisfying $(t_1, t_2) \vDash$ CMD$_1$, they must satisfy $(t_1, t_2) \vDash$ CMD$_2$ as well. Again, we illustrate the conclusion by two possible cases:

- if $(t_1, t_2) \not\asymp$ [name $\approx$ Joy Alice] $\wedge$ [address $\approx *$], then $(t_1, t_2) \vDash$ CMD$_2$, since its left-hand-side is not agreed;
- if $(t_1, t_2) \asymp$ [name $\approx$ Joy Alice] $\wedge$ [address $\approx *$], we have $(t_1, t_2) \asymp$ [name $\approx$ Alice] $\wedge$ [address $\approx *$] as well, according to Alice $\succeq$ Joy Alice. It implies $(t_1, t_2) \asymp$ [ssn $\rightleftharpoons *$] by CMD$_1$. In other words, (t_1, t_2) satisfy CMD$_2$ too;

The above augmentation and left-dominating rules are practically important, especially in the following discovery problem. We defer reporting other inference rules to form a complete and sound inference system as the future work. In fact, the implication problem of dependencies with conditions is high non-trivial. For example, the CFDs implication has been proved co-NP-complete [11], where similarity metrics are not considered. Interesting future work including the consistency and implication problems for CMDs are discussed in Section 6.

Reducing CMDs

According to logical implication, there may exist redundancy in an arbitrary set of CMDs, that is, those ones can be implied by others. In order to reduce such redundancy, it is natural to explore a concise set of CMDs that cannot be implied with each other. However, the hardness of implication has been mentioned above. Therefore, instead of finding a minimal cover of CMDs which can imply all the others with least redundancy, we study those CMDs that are irreducible, namely irreducible CMDs.

Specifically, if a CMD_1 implies another CMD_2 by augmentation and left-dominating, we say that CMD_2 is reducible to CMD_1.

DEFINITION 3.1. *Given any instance I, a* CMD *having $I \vDash [X \approx x] \rightarrow [Y \rightleftharpoons y]$ is* irreducible*, if there does not exist any $z \in \mathsf{dom}(Z)$ such that $Z \subseteq X$, $z \succeq x[Z]$ and $I \vDash [Z \approx z] \rightarrow [Y \rightleftharpoons y]$.*

Here, $y \in \mathsf{dom}(Y)$ could also be $*$. Suppose that such $[Z \approx z]$ exists which can also determine $[Y \rightleftharpoons y]$. Then, $[X \approx x] \rightarrow [Y \rightleftharpoons y]$ is implied by $[Z \approx z] \rightarrow [Y \rightleftharpoons y]$ according to the augmentation and left-dominating in Lemma 3.1 and 3.2, i.e., reducible. For instance, CMD_2 in Example 3.1 is not an irreducible CMD, since we have $\mathsf{Alice} \succeq \mathsf{Joy\ Alice}$ with respect to the domain of name. That is, CMD_2 can be implied by CMD_1 and is reducible. Moreover, CMD_3, whose left-hand-side attributes are a superset of CMD_1, is reducible too according to the irreducible definition.

Given any instance I, there exist a set Σ of all CMDs that hold in I. Now we can find a set $\Sigma_c \subseteq \Sigma$ of all irreducible CMDs that can imply the other CMDs in Σ by augmentation and left-dominating. It is notable that a set of irreducible CMDs is different from a minimal cover for all CMDs, which relies on a complete and sound inference system. A minimal cover Σ_m denotes a minimal set of CMDs which can imply all the others, while a set Σ_c consists of all irreducible CMDs. For example, let $\Sigma = \{\text{CMD}_1, \text{CMD}_2, \text{CMD}_3\}$ in Example 3.1 be the set of all CMDs holding in a relation instance I. Then, a set Σ_c of irreducible CMDs with respect to Σ can be $\{\text{CMD}_1\}$, where CMD_1 is irreducible and can imply all the other CMD_2 and CMD_3 by left-dominating and augmentation, respectively.

An irreducible MD must also be an irreducible CMD. The rationale is that all the MDs can be interpreted as special cases of CMDs. Given any instance I, let Σ_c be a set of all irreducible CMDs and Γ_c be a set of all minimal MDs that hold in I. We always have $\Sigma_c \supseteq \Gamma_c$.

3.2 Discovering Irreducible CMDs

Note that CMDs might hardly be given directly by user's domain knowledge in most cases, especially to specify those specific conditions. In practice, a set of sample data, say I, as truth can often be provided. Possible dependency semantics are embedded in the truth data, i.e., $I \vDash \Sigma$. Thus, in the following of this section, we focus on discovering CMDs from the sample data.

Formally, we study the discovery of irreducible CMDs that hold in a sample data instance, to avoid redundancy.

PROBLEM 3.1. *Given an instance I of relation R and a target $[Y \rightleftharpoons y]$ of matching rules, it is to find a set Σ for all irreducible* CMDs*, which determine $[Y \rightleftharpoons y]$.*

Recall that all the attribute values including the virtual value $*$ can be possible conditions. Let m be the number of attributes in $R \backslash Y$. The search space of candidate conditions for CMDs is

$$\mathsf{dom}(A_1) \times \mathsf{dom}(A_2) \times \cdots \times \mathsf{dom}(A_m), A_i \in R \setminus Y$$

Let D be the average size of $\mathsf{dom}(A_i)$ of each attribute. We have the search space complexity $O(D^m)$.

Straight-forward Algorithm

A straight-forward approach is to evaluate all the possible candidates of the search space. Let Σ be the set of discovered CMDs, initially having $\Sigma = \emptyset$. If a CMD from the search space holds in the given instance I, then we add it into Σ. Finally, we remove those CMDs in Σ that are reducible.

Let n be the number of tuples in the given relation instance, $|I|$. The evaluation of a CMD in I (hold or not) can be done in $O(n^2)$ time. We have a search space with size $O(D^m)$. Therefore, the time complexity of straight-forward approach is $O(n^2 D^m)$. Unfortunately, as illustrated in the following, the discovery problem is intrinsically hard. It is well known that a minimal cover of all functional dependencies discovered from a sample data can be exponentially large in size with respect to the number of attributes [25]. Since our set of all irreducible CMDs may not be minimal, that is, may be larger than minimal cover, it is not surprising that the exponential size carries over to irreducible CMDs. Due to this inherent hardness in finding all irreducible CMDs in an instance I of relation R, in the following, we focus on pruning techniques that may improve the discovery efficiency in practice.

Domain-oriented Algorithm

Recall that, given any instance I, if a CMD is found to hold in I, we can directly tell some other CMDs also holding in I according to the logical implication in Section 3.1. Similar idea is also adopted in discovering FDs and CFDs [7, 12], e.g., TANE [19] applies the pruning between levels of attribute sets according to the augmentation rule. Besides the augmentation, we can further utilize the left-dominating inference rule for pruning CMDs. Note that the dominating relationships between domain values are not considered in previous work, e.g., the CFDs discovery [7, 12] only needs to consider the relationship with respect to the virtual value. Thereby, we develop the following domain-oriented algorithm, to handle the order relation of domain values in different attributes respectively.

According to the augmentation property in Lemma 3.1, for any CMD with left-hand-side attributes X, the corresponding augmentation CMDs on supper set of X might be implied. Therefore, similar to the level-wise algorithm for discovering FDs [19, 20], we can also consider the left-hand-side attributes incrementally, i.e., traverse the attributes from a smaller attribute set to larger ones.

Let $X \subseteq R \setminus Y$ be the current set of left-hand-side attributes, as illustrated in Figure 2. Let Σ_X be the set of CMDs whose left-hand-side attributes are subsets of X and hold in I, and $\bar{\Sigma}_X$ be the set of CMDs that do not hold in I, where $\Sigma_X \cup \bar{\Sigma}_X$ exactly corresponds to the search space defined on X.

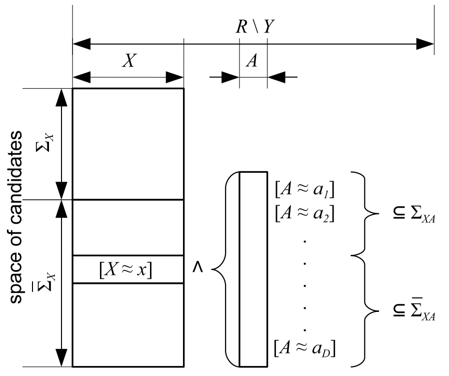


Figure 2: Domain-oriented pruning of candidates

We consider a remaining attribute $A \in R \setminus (X \cup Y)$. According to the augmentation property in Lemma 3.1, for any $[X \approx x] \to [Y \rightleftharpoons y] \in \Sigma_X$, the corresponding $[X \approx x] \wedge [A \approx a] \to [Y \rightleftharpoons y]$ must hold in I as well and is reducible. For example, if we find CMD$_1$ in Example 3.1 holding in I, then CMD$_3$ with augmentation $[\mathsf{institute} \approx *]$ must also hold and can be directly ignored without evaluation in I since it is reducible. In other words, we only have to consider the candidates with augmentation on CMDs in $\bar{\Sigma}_X$.

Let $[X \approx x] \to [Y \rightleftharpoons y] \in \bar{\Sigma}_X$ be a CMD that does not hold in I. Suppose that its augmentation with $[A \approx a]$ holds in I based on the evaluation, i.e., $I \vDash [X \approx x] \wedge [A \approx a] \to [Y \rightleftharpoons y]$. Then, according to Lemma 3.2, the augmentation with any $[A \approx b], a \succeq b, b \in \mathsf{dom}(A)$, must hold in I as well and is reducible. In other words, such augmentation with $[A \approx b]$ can be safely pruned without evaluation in I. For example, if we find CMD$_1$ in Example 3.1 holding in I, then CMD$_2$, whose left-hand-side condition is dominated by that of CMD$_1$, must also hold and can be directly pruned without evaluation in I since it is reducible.

Algorithm 1 presents the procedure of domain-oriented discovery, where $\bar{\Sigma}$ denotes the set of CMDs that do not hold in I with respect to current X in each iteration. To avoid reducible case, for the values in $\mathsf{dom}(A)$ of each attribute A, we consider an ordering based on the dominating relationship. That is, for any i-th value $a_i \in \mathsf{dom}(A)$ and j-th value $a_j \in \mathsf{dom}(A)$, $i < j$, we have either $a_i \succeq a_j$ or no dominating relationship between a_i, a_j. In other words, we always first process those values dominating others. Thereby, the results Σ_A in each iteration should be irreducible. Finally, the operator $\uplus$ combines previous results Σ together with current ones Σ_A and removes reducible CMDs.

Let r be the pruning rate on average in each step, that is, rD values of an attribute A can be pruned on average. Then, Algorithm 1 runs in $O(n^2(1-r)^m D^m)$ time. Although, the time complexity is still exponential with respect to the size of relation schema $|R|$, in practice, the domain-oriented pruning algorithm shows significantly lower time cost than straight-forward approach.

```
Procedure IRREDUCIBLE($[Y \rightleftharpoons y], I$)
Input: A target $[Y \rightleftharpoons y]$ and an instance $I$.
Output: A set $\Sigma$ of irreducible CMDs determining $[Y \rightleftharpoons y]$
    in $I$.
 1: $\Sigma := \emptyset, X := \emptyset, \bar{\Sigma} := \{[X \approx *] \to [Y \rightleftharpoons y]\}$
 2: for each $A \in R \setminus (X \cup Y)$ do
 3:     $\Sigma_A := \emptyset, \bar{\Sigma}_A := \emptyset$
 4:     for each $[X \approx x] \to [Y \rightleftharpoons y] \in \bar{\Sigma}$ do
 5:         for each $a \in \mathsf{dom}(A)$ do
 6:             $\bar{\Sigma}_A := \bar{\Sigma}_A \cup \{[A \approx a] \wedge [X \approx x] \to [Y \rightleftharpoons y]\}$
 7:     for each $[W \approx w] \to [Y \rightleftharpoons y] \in \bar{\Sigma}_A$ do
 8:         if $I \vDash [W \approx w] \to [Y \rightleftharpoons y]$ then
 9:             $\Sigma_A := \Sigma_A \cup \{[W \approx w] \to [Y \rightleftharpoons y]\}$
10:             remove $[W \approx w] \to [Y \rightleftharpoons y]$ from $\bar{\Sigma}_A$
11:             remove all $[W \approx v] \to [Y \rightleftharpoons y]$ from $\bar{\Sigma}_A$ such
                that $w \succeq v$
12:     $X := X \cup \{A\}$
13:     $\Sigma := \Sigma \uplus \Sigma_A, \bar{\Sigma} := \bar{\Sigma}_A$
14: return $\Sigma$
```

Algorithm 1: Domain-oriented pruning

Tuple-oriented Algorithm

In the above domain-oriented algorithm, during the evaluation of each CMD, i.e., whether $I \vDash [W \approx w] \to [Y \rightleftharpoons y]$, the entire instance I has to be traversed. However, according to the implication relationship of CMDs, the set of tuples in I agreeing a CMD can be a superset/subset of those tuples agreeing another CMD. Such idea is applied as a depth-first strategy for discovering FDs (FastFD [30]) and CFDs (FastCFD [12]). Thereby, in the following, we present a tuple-oriented algorithm by avoid traversing all the tuples in I in each evaluation. Again, we have to address the dominating relationships between different values which are not studied previously.

Given an instance I, it specifies a set of tuples that can determine $[Y \rightleftharpoons y]$, or equivalently, it also specifies a set of tuples that do not agree $[Y \rightleftharpoons y]$. Recall that each CMD specifies dependency constraint on certain subset of tuples in I which agree $[Y \rightleftharpoons y]$, while the other tuples not agreeing $[Y \rightleftharpoons y]$ are excluded by conditions $[X \approx x]$ in CMD. Intuitively, the tuple-oriented approach targets on the CMDs which can exclude all these tuples not agreeing $[Y \rightleftharpoons y]$.

First, we formally introduce the exclusion of tuples by conditions. Let Υ be a set of distinct tuple pairs from I,

$$\Upsilon = \{(t_i, t_j) \mid t_i, t_j \in I\}.$$

Let $[X \approx x]$ be any condition. We define a selection operator σ on Υ with respect to exclusion, for example,

$$\sigma(\Upsilon, \neg[X \approx x]) = \{(t_i, t_j) \in \Upsilon \mid (t_i, t_j) \nasymp [X \approx x]\}.$$

where $(t_1, t_2) \nasymp [X \approx x]$ denotes that t_1, t_2 do not agree the condition $[X \approx x]$, i.e., either $t_1[X] \not\approx t_2[X]$ or $t_1[X] \not\approx x$ or $t_2[X] \not\approx x$. It denotes all the tuple pairs that are excluded by $[X \approx x]$, that is, not agreeing the condition.

This operator can also be applied on $[Y \rightleftharpoons y]$ as well,

$$\sigma(\Upsilon, \neg[Y \rightleftharpoons y]) = \{(t_i, t_j) \in \Upsilon \mid (t_i, t_j) \nasymp [Y \rightleftharpoons y]\}.$$

As illustrated in Figure 3, by applying different selection conditions, the tuple pairs in Υ can be divided into four partitions.

LEMMA 3.3. *A* CMD *holds in an instance,* $I \vDash [X \approx x] \rightarrow [Y \rightleftharpoons y]$, iff

$$\sigma(\Upsilon_I, \neg[X \approx x]) \supseteq \sigma(\Upsilon_I, \neg[Y \rightleftharpoons y]),$$

or equivalently,

$$\sigma(\Upsilon_I, [X \approx x] \wedge \neg[Y \rightleftharpoons y]) = \emptyset,$$

where Υ_I *denotes the set of all distinct tuples pairs from* I.

In other words, to find an irreducible CMD is exactly to find an irreducible $[X \approx x]$ such that there does not exist any tuple pair in Υ_I agreeing $[X \approx x]$ but not $[Y \rightleftharpoons y]$, that is, to exclude all the tuples pairs $\neg[Y \rightleftharpoons y]$ by $[X \approx x]$.

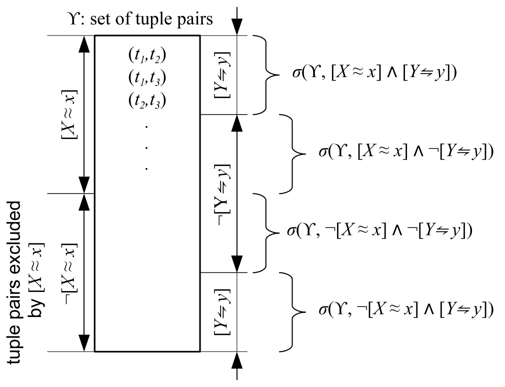


Figure 3: Tuple exclusion by conditions of a cmd

Next, in order to avoid traversing all the tuple pairs in Υ when evaluating different conditions, we study their corresponding selection results of tuple pairs in Υ. Consider any Υ,

LEMMA 3.4. *For any* $x_1 \succeq x_2, x_1, x_2 \in \mathsf{dom}(X)$, *we have*

$$\sigma(\Upsilon, [X \approx x_1]) \supseteq \sigma(\Upsilon, [X \approx x_2]),$$

or equivalently,

$$\sigma(\Upsilon, \neg[X \approx x_1]) \subseteq \sigma(\Upsilon, \neg[X \approx x_2]).$$

Let $\Upsilon' = \sigma(\Upsilon, [X \approx x_1])$. According to the lemma, we have $\Upsilon' \supseteq \sigma(\Upsilon, [X \approx x_2]) = \sigma(\Upsilon', [X \approx x_2])$. That is, we can use the subset Υ' of tuple pairs for evaluating $[X \approx x_2]$, instead of the entire Υ.

LEMMA 3.5. *Let* $[V \approx v] = [X \approx x] \wedge [A \approx a]$, *we have*

$$\sigma(\Upsilon, [X \approx x]) \supseteq \sigma(\Upsilon, [V \approx v]),$$

or equivalently,

$$\sigma(\Upsilon, \neg[X \approx x]) \subseteq \sigma(\Upsilon, \neg[V \approx v]).$$

Similarly, let $\Upsilon' = \sigma(\Upsilon, [X \approx x])$. According to the lemma, we can use the subset Υ' of tuple pairs for evaluating $[V \approx v]$, instead of the entire Υ.

Finally, Algorithm 2 presents the pseudo-code for tuple-oriented exclusion. Instead of the entire instance I, the algorithm uses Υ of tuple pairs with respect to I as input. The left-hand-side attributes X increase from small to large

```
Procedure IRREDUCIBLE($\Upsilon, [X \approx x], Z$)
Input: A set $\Upsilon$ of tuple pairs from $I$ and a condition $[X \approx x]$.
Output: A set $\Sigma$ of irreducible CMDs determining $[Y \rightleftharpoons y]$ in $I$.
 1: if $Z$ is empty then
 2:    return $\emptyset$
 3: $A :=$ an attribute removed from $Z$
 4: $\Sigma := $ IRREDUCIBLE($\Upsilon, [X \approx x], Z$)
 5: $\mathsf{dom}_{\mathsf{local}}(A) :=$ dynamic computed from $\Upsilon$
 6: for each $a \in \mathsf{dom}_{\mathsf{local}}(A)$ do
 7:    $\Upsilon' := \sigma(\Upsilon, [A \approx a])$
 8:    if $\Upsilon' = \emptyset$ then
 9:       $\Sigma := \Sigma \uplus \{[X \approx x] \wedge [A \approx a] \rightarrow [Y \rightleftharpoons y]\}$
10:       remove all $b \in \mathsf{dom}_{\mathsf{local}}(A)$ such that $a \succeq b$
11:    else
12:       $\Sigma := \Sigma \uplus$ IRREDUCIBLE($\Upsilon', [X \approx x] \wedge [A \approx a], Z$)
13: return $\Sigma$
```

Algorithm 2: Tuple-oriented exclusion

in each recursion, in order to utilize the subset relationship of Υ in Lemma 3.5. Initially, we have $\Upsilon = \sigma(\Upsilon_I, \neg[Y \rightleftharpoons y])$, $[X \approx x] =$null and $Z = R \setminus Y$.

In each recursion, the current Υ records tuple pairs that agree the current $[X \approx x]$ but not $[Y \rightleftharpoons y]$, i.e., $\Upsilon = \sigma(\Upsilon_I, [X \approx x] \wedge \neg[Y \rightleftharpoons y])$. Let A be the attribute considered in the current recursion. For each $[A \approx a]$, according to Lemma 3.5, we can compute a

$$\Upsilon' = \sigma(\Upsilon, [A \approx a]) = \sigma(\Upsilon_I, [X \approx x] \wedge [A \approx a] \wedge \neg[Y \rightleftharpoons y]).$$

As stated in Lemma 3.3, if $\Upsilon' = \emptyset$, then the CMD holds in I.

As mentioned, in each recursion, we evaluate a subset Υ of tuple pairs instead of the entire I. Therefore, other than considering the full $\mathsf{dom}(A)$ defined on R, we dynamically compute a local version with respect to the current Υ, $\mathsf{dom}_{\mathsf{local}}(A) = \{a \in \mathsf{dom}(A) \mid \exists(t_i, t_j) \in \Upsilon, (t_i, t_j) \asymp [A \approx a]\}$. Once a CMD is found to hold, similar to the pruning of conditions in domain-oriented algorithm, we can also prune the remaining dominated conditions according to Lemma 3.4, i.e., line 10 in Algorithm 2.

Let l be the exclusion rate on average in each recursion, that is, $(1-l)|\Upsilon|$ tuple pairs remained after each exclusion. Note that the $\mathsf{dom}_{\mathsf{local}}(A)$ is no greater than $\mathsf{dom}(A)$. In other words, the complexity of tuple-oriented exclusion should be no worse than that of domain-oriented algorithm. The tuple-oriented algorithm runs in $O(n^2(1-l)^m D^m (1-r)^m)$ time.

Comparison

In summary, the straight-forward approach evaluate all candidates in the instance I to find all irreducible CMDs. The domain-oriented approach also evaluates the entire instance I and studies the pruning of candidates in a static domain. The tuple-oriented approach incrementally excludes tuple pairs with respect to I and introduces dynamic domain according to the currently remaining tuple pairs.

4. EXPERIMENTS

In this section, we report an extensive experimental evaluation. The experiments on real data sets mainly cover two aspects, the performance of discovering CMDs. All the algo-

Table 2: Examples of discovered CMDs

CMD$_1$	[title $\approx *$] $\wedge$ [venue $\approx$ journal of computer and system sciences] $\rightarrow$ [id $\rightleftharpoons *$]
CMD$_2$	[title $\approx$ a general lower bound on the number of examples needed for learning] $\wedge$ [venue $\approx *$] $\rightarrow$ [id $\rightleftharpoons *$]
CMD$_3$	[title $\approx$ a decision theoretic generalization of on line learning and an ...] $\wedge$ [venue $\approx *$] $\rightarrow$ [id $\rightleftharpoons *$]
CMD$_4$	[author $\approx *$] $\wedge$ [title $\approx *$] $\wedge$ [venue $\approx *$] $\rightarrow$ [id $\rightleftharpoons *$]
CMD$_5$	[author $\approx *$] $\wedge$ [title $\approx$ bounds on the sample complexity of bayesian learning using ...] $\rightarrow$ [id $\rightleftharpoons *$]
CMD$_6$	[author $\approx$ a. ehrenfeucht, d. haussler, m. kearns, and l. valiant.] $\wedge$ [venue $\approx *$] $\rightarrow$ [id $\rightleftharpoons *$]
CMD$_7$	[author $\approx$ freund, y. and schapire, r. e.] $\wedge$ [venue $\approx *$] $\rightarrow$ [id $\rightleftharpoons *$]
CMD$_8$	[author $\approx$ freund, y. and schapire, r. e.] $\wedge$ [title $\approx *$] $\rightarrow$ [id $\rightleftharpoons *$]
CMD$_9$	[title $\approx *$] $\wedge$ [venue $\approx$ 33 proceedings of the 1988 workshop on ...] $\wedge$ [address $\approx$ boston, ma] $\rightarrow$ [id $\rightleftharpoons$ freund0000a]

rithms are implemented by Java and run on a machine with Intel Core 2 CPU (2.13 GHz) and 2 GB of memory.

Data Sets

Two real data sets are adopted in the experiments, i.e., Restaurant[2] includes records of restaurants with schema

$$(\mathsf{name}, \mathsf{address}, \mathsf{city}, \mathsf{type}, \mathsf{id}).$$

Cora[3] collects scientific research paper citations with schema

$$(\mathsf{author}, \mathsf{volume}, \mathsf{title}, \mathsf{institution}, \mathsf{venue}, \mathsf{address}, \mathsf{year}, \dots, \mathsf{id}).$$

Attribute id indicates a manually labeled identification. For example, two tuples in Restaurant with identical id means that they are the same restaurant in real world. Thus, the id attribute is adopted as the right-hand-side attribute Y in the following experiments. For the similarity operator $\approx$ on the remaining attributes, we use cosine similarity on q-grams [9] with threshold 0.8.

Discovery Evaluation

The first experiment illustrates some example results found in the data sets. Table 2 shows several irreducible CMDs discovered in Cora. For example, the first CMD$_1$ states that: for any two tuples appearing in the venue similar to journal of computer and system sciences, if their title are similar as well, then these two tuples must describe the same paper having identical id. As special cases, our approach can find all the MDs as well, for instance, CMD$_4$ holds in the entire table without any condition, i.e., a traditional MD. The results in Table 2 also verifies that returned CMDs are irreducible. That is, CMDs that can be implied by the results according to augmentation and left-dominating will not appear. With the same condition [author $\approx$ freund, y. and schapire, r. e.], CMD$_7$ specifies a dependency with respect to [venue $\approx *$], while CMD$_8$ denotes another different case of similar title.

To evaluate the performance of discovering CMDs, we conduct discovery approaches in various settings of relation schema R and instance I. Three proposed discovery algorithms, including straight-forward (SF), domain-oriented pruning (DOP) and tuple-oriented exclusion (TOE), are compared. It is worth noting that these three approaches are exact algorithms, i.e., they discover the same set of dependencies.

First, we evaluate in instances from I_1 to I_6 with number of tuples from 100 to 600, respectively. The results are reported in sub-figure (a) of Figure 4 and 5. As presented, the advanced DOP and TOE can significantly improve the time

[2]http://www.cs.utexas.edu/users/ml/riddle/data.html
[3]http://www.cs.umass.edu/~mccallum/code-data.html

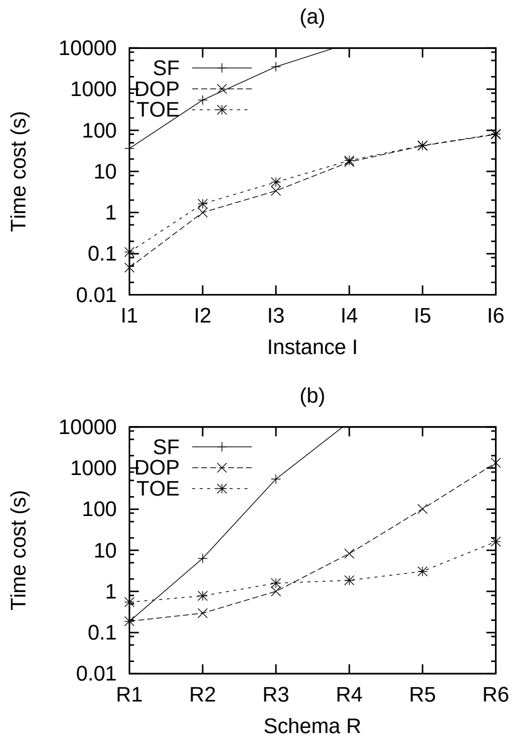
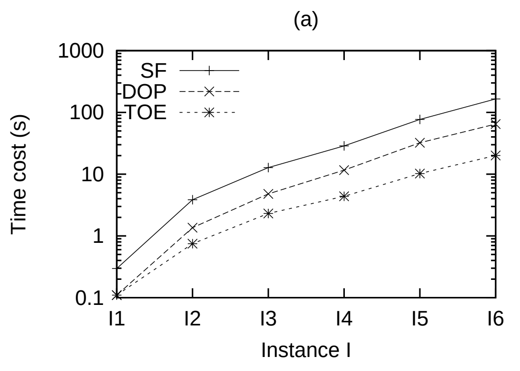


Figure 4: Discovery performance in Cora

performance compared with SF approach. Note that the average domain size of attributes in Cora is larger than that of Restaurant. Thus, the improvement achieved by DOP and TOE in Cora data set is greater as well, i.e., about 3 orders of magnitude. Moreover, the time cost of TOE increases a bit slower than the increase of DOP with respect to the instance size I. It is because the tuple-oriented approach (TOE) does not have to traverse the entire instance in each evaluation.

Next, we study the discovery on various relation schema R. For the Cora data set, relation schemas R_1 to R_6 have attributes from 2 to 7. Since the Restaurant data set has only 5 attributes, as presented in Table 3, the schemas are prepared in different attribute sets with various domain sizes. For example, in R_2, the domain size of attribute name is 106 and the domain size of address is 22. It implies the domain size of $|\mathsf{dom}(\mathsf{name}, \mathsf{address})| = 2332$.

As illustrated in sub-figure (b) of Figure 4 and 5, the discovery time cost increases exponentially with the increase of schema size. As discussed, the search space depends on



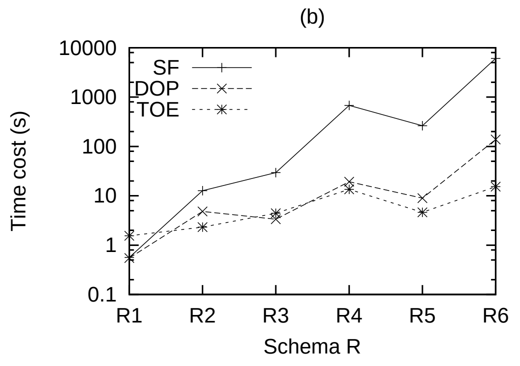


Figure 5: Discovery performance in Restaurant

Table 3: Restaurant schemas

	$R \setminus Y$	$\lvert\mathsf{dom}(R \setminus Y)\rvert$	
R_1	name	106	=106
R_2	name, address	106*22	=2332
R_3	name, city	106*52	=5512
R_4	name, address, city	106*22*52	=121264
R_5	name, city, type	106*52*8	=44096
R_6	name, address, city, type	106*22*52*8	=970112

the size of $\lvert\mathsf{dom}(R \setminus Y)\rvert$. Thus, as presented in Figure 5 (b), the time cost is approximately proportional to the corresponding domain size in Table 3. Similar results are also observed in Figure 4 (b), which increase exponentially with respect to the number of attributes. Compared with the straight-forward one, the advanced DOP and TOE approaches can achieve several orders of magnitude improvement. For example, the TOE achieves an improvement of 4 orders of magnitude compared with SF one in R_4 of Cora. And most importantly, the tuple-oriented approach TOE scales much better than DOP. When the schema size is small, e.g., R_1 of Cora or Restaurant, the domain-oriented algorithm DOP does not need to maintain a certain set Υ of tuple pairs, and thus shows lower time cost. On the other hand, if the schema size is large, e.g., R_6 of Cora, TOE shows 2 orders of magnitude lower time cost than that of DOP. Similar result is also observed in R_6 of Restaurant. This result also verifies our complexity analysis for DOP and TOE with respect to schema size.

5. RELATED WORK

Recently, traditional dependencies, such as functional dependencies (FDs) and inclusion dependencies (INDs) for the schema design [1], are revisited for new applications like improving the quality of data. The conditional functional dependencies (CFDs), as an extension of traditional FDs with conditions, are first proposed in [3] for data cleaning. The basic idea of CFDs is making the FDs, originally hold for the whole table, valid only for a set of tuples specified by the conditions. Cong et al. [8] study the detecting and repairing methods of violation by CFDs. Fan et al. [14] investigate the propagation of CFDs for data integration. Bravo et al. [4] propose an extension of CFDs by employing disjunction and negation. Golab et al. [17] define a range tableau for CFDs, where each value is a range. In addition, Bravo et al. [5] propose conditional inclusion dependency (CINDs), which are useful not only in data cleaning, but are also in contextual schema matching. Instead of associating conditions based on identification, in this study, we investigate conditions with respect to similarity matching quality.

Matching dependencies (MDs) are first proposed in [10] for specifying matching rules for the object identification (see [9] for a survey). The MDs can be regarded as a generalization of FDs, which are based on identical values having similarity equal to 1.0 exactly. Thus, FDs can be represented by the syntax of MDs as well. Reasoning mechanism for deducing MDs from a set of given MDs is studied in [13]. As illustrated, MDs and their reasoning techniques can improve the effectiveness of record matching. Besides matching dependencies, in recent work, the importance of introducing similarity metrics in dependencies has been commonly recognized. Koudas et al. [23] study the dependencies with similarity metrics on attributes Y when given the identification on X. Golab et al. [16] propose sequential dependencies, which also targets on the metric distance/similarity of values. A sequential dependency, in the form of $X \rightarrow_g Y$, states that when tuples are sorted on X, the distance between the Y-values of any two consecutive tuples are within interval g.

The discovery of dependencies from a given relation instance is widely studied [2, 26, 27, 28, 24]. In discovering FDs, previous work targeted on generating a minimal cover of all FDs. Huhtala et al. [19, 20] propose a levelwise algorithm, namely TANE, together with efficient pruning when searching in the lattice of attributes. Remarkably, TANE algorithm also supports the discovery of approximate FDs. Wyss et al. [30] study depth-first, heuristic-driven algorithm, namely FastFDs, which is (almost) linear to the size of FDs cover. Flach and Savnik [15] discover FDs in a bottom-up style, which considers the maximal invalid dependencies first. When searching in a hypotheses space, the maximum invalid dependencies are used for pruning the search space. In discovering CFDs, Chiang and Miller [7] explore CFDs by considering all the possible dependency rules when $X \rightarrow Y$ is not specified. In [12], Fan et al. also study the case when the embedded FDs are not given, and propose three algorithms for different scenarios. When a rule $X \rightarrow Y$ is suggested, Golab et al. [17] study the discovery of optimal CFDs with the minimum pattern tableau size. A concise set of patterns are naturally desirable which may have lower cost during the applications such as violation detection by CFDs. Unfortunately, similarity metrics of MDs and CMDs are not considered in the previous work of discovering dependencies.

6. CONCLUSIONS

In this study, we propose conditional matching dependencies. To out best knowledge, this is the first work on matching dependencies with conditions. A comprehensive syntax system is developed with formal definitions of conditional matching dependencies and dominating relationships among conditions. Inference rules including augmentation and left-dominating are then introduced, which raises the problem of finding irreducible CMDs. Due to the intrinsical hardness of discovering CMDs, we develop several pruning algorithms to improve discovery performance in practice. Finally, an extensive experimental evaluation is reported, including the performance of discovering CMDs.

Besides the problems addressed in this study, i.e., discovery of CMDs, there are many other aspects to explore in future work. For example, the consistency problem of CMDs is: given any set Σ of CMDs to decide whether there exists a relation instance I such that $I \vDash \Sigma$. It is not only theoretically interesting but also practically useful, e.g., in data cleaning practice, the input constraint rules of CMDs should not be self-contradictory. Another interesting problem is the complexity of implication problem together with a complete and sound inference system for CMDs. It is already known that implication of CFDs is co-NP-complete, where similarity metrics are not considered. The implication analysis for CMDs appears non-trivial as well.

Acknowledgments

This work is supported by HONG KONG RGC GRF grant under project no. 611608.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] D. Bitton, J. Millman, and S. Torgersen. A feasibility and performance study of dependency inference. In *ICDE*, pages 635–641, 1989.

[3] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.

[4] L. Bravo, W. Fan, F. Geerts, and S. Ma. Increasing the expressivity of conditional functional dependencies without extra complexity. In *ICDE*, pages 516–525, 2008.

[5] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. In *VLDB*, pages 243–254, 2007.

[6] S. Chaudhuri, A. D. Sarma, V. Ganti, and R. Kaushik. Leveraging aggregate constraints for deduplication. In *SIGMOD Conference*, pages 437–448, 2007.

[7] F. Chiang and R. J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.

[8] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, pages 315–326, 2007.

[9] 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.

[10] W. Fan. Dependencies revisited for improving data quality. In *PODS*, pages 159–170, 2008.

[11] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.*, 33(2), 2008.

[12] W. Fan, F. Geerts, L. V. S. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. In *ICDE*, pages 1231–1234, 2009.

[13] W. Fan, J. Li, X. Jia, and S. Ma. Reasoning about record matching rules. *PVLDB*, 2009.

[14] W. Fan, S. Ma, Y. Hu, J. Liu, and Y. Wu. Propagating functional dependencies with conditions. *PVLDB*, 1(1):391–407, 2008.

[15] P. A. Flach and I. Savnik. Database dependency discovery: A machine learning approach. *AI Commun.*, 12(3):139–160, 1999.

[16] L. Golab, H. J. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1):574–585, 2009.

[17] L. Golab, H. J. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376–390, 2008.

[18] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD Conference*, pages 127–138, 1995.

[19] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *ICDE*, pages 392–401, 1998.

[20] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[21] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[22] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theor. Comput. Sci.*, 149(1):129–149, 1995.

[23] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *ICDE*, pages 1275–1278, 2009.

[24] S. Kramer and B. Pfahringer. Efficient search for strong partial determinations. In *KDD*, pages 371–374, 1996.

[25] H. Mannila and K.-J. Räihä. Dependency inference. In *VLDB*, pages 155–158, 1987.

[26] H. Mannila and K.-J. Räihä. *Design of Relational Databases*. Addison-Wesley, 1992.

[27] H. Mannila and K.-J. Räihä. Algorithms for inferring functional dependencies from relations. *Data Knowl. Eng.*, 12(1):83–99, 1994.

[28] J. C. Schlimmer. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *ICML*, pages 284–290, 1993.

[29] W. Shen, X. Li, and A. Doan. Constraint-based entity matching. In *AAAI*, pages 862–867, 2005.

[30] C. M. Wyss, C. Giannella, and E. L. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In *DaWaK*, pages 101–110, 2001.