Uncertain data integration using functional dependencies

Ayat, S.N.; Afsarmanesh, H.; Akbarinia, R.; Valduriez, P.

Uncertain Data Integration Using Functional Dependencies

Naser Ayat[#1], Hamideh Afsarmanesh[#2], Reza Akbarinia[*3], Patrick Valduriez[*4]

[#]Informatics Institute, University of Amsterdam, Amsterdam, Netherlands
[1]s.n.ayat@uva.nl, [2]h.afsarmanesh@uva.nl
[*]INRIA and LIRMM, Montpellier, France
[3,4]{Firstname.Lastname@inria.fr}

Abstract. Data integration systems are crucial for applications that need to provide a uniform interface to a set of autonomous and heterogeneous data sources. However, setting up a full data integration system for many application contexts, e.g. web and scientific data management, requires significant human effort which prevents it from being really scalable. In this paper, we propose IFD (Integration based on Functional Dependencies), a pay-as-you-go data integration system that allows integrating a given set of data sources, as well as incrementally integrating additional sources. IFD takes advantage of the background knowledge implied within functional dependencies for matching the source schemas. Our system is built on a probabilistic data model that allows capturing the uncertainty in data integration systems. Our performance evaluation results show significant performance gains of our approach in terms of recall and precision compared to the baseline approaches. They confirm the importance of functional dependencies and also the contribution of using a probabilistic data model in improving the quality of schema matching. The analytical study and experiments show that IFD scales well.

Keywords: data integration, uncertain data integration, functional dependency

1 Introduction

Data integration systems offer uniform access to a set of autonomous and heterogeneous data sources. Sources may range from database tables to web sites, and their numbers can range from tens to thousands. The main building blocks of a typical data integration application are mediated schema definition, schema matching and schema mapping. The mediated schema is the schema on which users pose queries. Schema matching is the process of finding associations between the elements (often attributes or relations) of different schemas, e.g. a source schema and the mediated schema in the popular Local As View (LAV) approach [1]. Schema mapping (also referred to as semantic mapping) is the process of relating the attributes of source schemas to the mediated schema (sometimes using expressions in a mapping language). The output of schema matching is used as input to schema mapping algorithms [1].

Setting up a full data integration system with a manually designed mediated schema requires significant human effort (e.g. domain experts and database designers). For example, in the context of the web, we are faced with a lot of data sources, even in a particular domain like travel, which makes it impossible to manually integrate the data sources. On the other hand, there are many application contexts, e.g. web, scientific data management, and personal information management, which do not require full integration to provide useful services [2]. These applications need to start with a data integration application in a complete automatic setting for reducing human effort and development time and put more effort on improving it as needed. Let us present a motivating example from the scientific data management context.

Example 1. Consider a researcher who is interested in the less-known or yet unknown functions of the protein ABCC8 related to diabetes. While biological experiments are the ultimate means for verifying predicted functions, she must first discover and suggest such functions. For doing this, she should perform manual exploratory searches over numerous online sources. For example, she should consider both well-known databases such as EntrezGene, EntrezProtein and less-known databases of other research labs as well. In this case, spending too much money and time for setting up a full data integration system is not reasonable and even not feasible due to the large number of such databases. Having a data integration system with approximate answers can considerably save the time and reduce the research cost in this domain. It is sufficient to set up such a system in a complete automatic setting and spend more effort to improve it only if it is necessary. This recent setting, referred to by pay-as-you-go data integration, has attracted considerable attention, e.g. [3–6]. The ultimate goal of this setting is to reduce human burden, and thereby reduce the time and cost of data integration while providing sufficient integration [3].

Our goal is to build a data integration system in a pay-as-you-go setting. To capture the uncertainty arising during the matching process, we generate Probabilistic Mediated Schemas (PMSs) which have shown to be promising [7]. The idea behind PMSs is to have several mediated schemas, each one with a probability that indicates the closeness of the corresponding mediated schema to the ideal mediated schema.

The related work closest to ours is that of Sarma et al. [4] which Based on PMSs proposed UDI (Uncertain Data Integration), an uncertain data integration system. However, there are significant differences between UDI and our work. First, the execution cost of UDI's algorithm for generating mediated schemas is exponential, while ours is PTIME. Second, UDI may end up with an exponential number of mediated schemas with low probabilities attached to them, particularly if the system parameters are not adjusted carefully, but this is not the case for our work. Another main difference is that UDI may fail to capture some important attribute correlations, and thereby produce low quality answers. Let us clarify this by an example which is the same as the running example in [4].

Example 2. Consider the following schemas both describing people:
$S_1(name, hPhone, hAddr, oPhone, oAddr)$
$S_2(name, phone, address)$
In S_2, the attribute *phone* can either be a home phone number or an office phone number, and the attribute *address* can either be a home address or an office address.

An ideal data integration system should capture the correlation between *hPhone* and *hAddr* and also between *oPhone* and *oAddr*. Specifically, it must generate schemas which group the *address* and *hAddr* together if *phone* and *hPhone* are grouped together. Similarly it should group the *address* and *oAddr* together if *phone* and *oPhone* are grouped together. In other words either of the following schemas should be generated (we abbreviate *hPbone*, *oPhone*, *hAddr*, *oAddr* as hP, oP, hA, and oA respectively):
$M_1(\{name, name\}, \{phone, hP\}, \{oP\}, \{address, hA\}, \{oA\})$
$M_2(\{name, name\}, \{phone, oP\}, \{hP\}, \{address, oA\}, \{hA\})$

Although these schemas are generated by UDI, they are overwhelmed by schemas in which the attribute correlations are not respected. Thus, by producing a large number of schemas which can easily be exponential, the desirable schemas get a very low probability. This occurs because UDI does not consider attribute correlations. Most attribute correlations are expressed within Functional Dependencies (FDs). For example let F_1 and F_2 be the set of FDs of S_1 and S_2 respectively:
$F_1 = \{hPhone \rightarrow hAddr, oPhone \rightarrow oAddr\}$
$F_2 = \{phone \rightarrow address\}$
These FDs show the correlation between attributes. For example, $hPhone \rightarrow hAddr$ indicates that the two attributes *hPhone* and *hAddr* are correlated. Considering the pairs of FDs from different sources can help us extracting these correlations and achieving the goal of generating mediated schemas that represent these correlations. For example, the FD pair $phone \rightarrow address$ and $hPhone \rightarrow hAddr$ indicates that if we group *phone* and *hPhone* together, we should also group *address* and *hAddr* together, as well as the *oPhone* and *oAddr*.

The set of attribute correlations that cannot be represented in one mediated schema implies having several correct mediated schemas at the same time. In Example 2, we cannot discard any of the schemas M_1 and M_2 in favor of the other because we do not know whether *phone* represents home phone or office phone, or whether *address* represents home address or office address. Actually, we cannot determine the correct schema without the help of the domain expert. Thus, we keep both, as uncertain mediated schemas, and assign them a probability of correctness. We discuss the details of generating mediated schemas and assigning probabilities to them in section 3.2. Traditional data integration systems which build only one mediated schema cannot capture this kind of uncertainty.

In this paper, we propose IFD (Integration based on Functional Dependencies), a pay-as-you-go data integration system that takes into account attribute correlations by using functional dependencies. We use a probabilistic data model

that enables us to capture uncertainty in mediated schemas. IFD sets up the system by creating a set of PMSs automatically and lets the user improve them when necessary. It uses algorithms that scale well in the number of sources. We model the schema matching problem as a clustering problem with constraints. This allows us to generate mediated schemas using algorithms designed for the latter problem. In our approach, we build a custom distance function for representing the knowledge of attribute semantics which we extract from FDs. We also propose a new metric (i.e. FD-point) for ranking the generated mediated schemas in the clustering process, and selecting high quality ones. IFD allows integrating a given set of data sources, as well as incrementally integrating additional sources, without needing to restart the entire process. To validate our approach, we implemented IFD as well as baseline solutions. The performance evaluation results show significant performance gains of our approach in terms of recall and precision compared to the baseline approaches. They confirm the importance of FDs in improving the quality of generated mediated schemas.

The rest of the paper is organized as follows. In Section 2, we make our assumptions precise and define the problem. In Section 3, we propose IFD, and describe its architecture, components and algorithms. We also analyze the execution cost of IFD's algorithms. Section 4 describes our performance validation. Section 5 discusses related work, and Section 6 concludes.

2 Problem Definition

In this section, we first give our assumptions and some background about PMSs. Then, we state the problem we address in this paper.

For the applications which we consider (e.g., scientific data management), we assume the availability of functional dependencies for the attributes of sources. This is a reasonable assumption in the applications which we consider, in particular scientific applications, because the data source providers are willing to provide the full database design information, including functional dependencies. However, there are contexts such as the web in which functional dependencies are not available. For these applications, we can use one of the existing solutions, e.g. [8–11, 5] to derive functional dependencies from data. Obviously, by using functional dependencies, we can obtain the primary key. Another assumption, which we make for ease of presentation, is that the data model is relational.

Now, we define some basic concepts, e.g. functional dependencies and mediated schemas, and then state the problem addressed in this paper. Let S be a set of source schemas, say $S = \{S_1, \ldots, S_n\}$, where for each $S_i, i \in [1, n], S_i = \{a_{i,1}, \ldots, a_{i,l_i}\}$, such that $a_{i,1}, \ldots, a_{i,l_i}$ are the attributes of S_i. We denote the set of attributes in S_i by $att(S_i)$, and the set of all source attributes as A. That is $A = \cup_i att(S_i)$. For simplicity, we assume that S_i contains a single table. Let F be the set of functional dependencies of all source schemas, say $F = \{F_1, \ldots, F_n\}$. For each $S_i, i \in [1, n]$, let F_i be the set of functional dependencies among the attributes of S_i, i.e. $att(S_i)$, where each $fd_j, fd_j \in F_i$ is of the form $L_j \rightarrow R_j$ and $L_j \subseteq att(S_i), R_j \subseteq att(S_i)$. In every F_i, there is one fd of the form $L_p \rightarrow R_p$, where $R_p = att(S_i)$, i.e. L_p is the primary key of S_i.

We assume that every attribute in the data sources can be matched with at most one attribute in other data sources, which means we only consider one-to-one mappings. We do this for simplicity and also because this kind of mapping is more common in practice. For a set of sources S, we denote by $M = \{A_1, \ldots, A_m\}$ a mediated schema, where $A_i \subseteq A$, and for each $i, j \in [1, m], i \neq j \Rightarrow A_i \cap A_j = \emptyset$. Each attribute involved in A_i is called a mediated attribute. Every mediated attribute ideally consists of source attributes with the same semantics.

While traditional database integration approaches rely only on one mediated schema, a probabilistic approach considers several mediated schemas each with a probability that indicates the likelihood of closeness of the mediated schema to the manually created mediated schema. A formal definition of probabilistic mediated schemas is as follows.

A probabilistic mediated schema (PMS) for a set S of source schemas is the set $N = \{(M_1, P(M_1)), \ldots, (M_k, P(M_k))\}$ where

- M_i is a mediated schema for S, where $i \in [1, k]$.
- For each $i, j \in [1, k], i \neq j \Rightarrow M_i \neq M_j$,i.e. M_i, M_j are different clusterings of $att(S)$.
- $P(M_i) \in (0, 1]$.
- $\sum_{i=1}^{k} P(M_i) = 1$.

Since each mediated schema corresponds to a clustering of source attributes, we can measure its quality by computing the F-measure of the clustering.

Let us now state the problem we address. Suppose we are given a set of source schemas S, and a set of functional dependencies F and a positive integer number k as input. Our problem is to efficiently find a set of k probabilistic mediated schemas which have the highest F-measure.

3 Data Integration Based on Functional Dependencies

Setting up a data integration application requires significant human effort particularly in creating the mediated schema and in generating the mappings between the mediated schema and the data sources [4]. Both activities require experts with good knowledge of the domain. In this section, we describe IFD, a data integration system that automatically performs the tasks of mediated schema generation and the attribute matching, by taking advantage of functional dependencies among the source attributes.

In the rest of this section, we first briefly describe the architecture of our data integration system. Then, we describe our approach for schema matching.

3.1 System Architecture

Figure 1 depicts the architecture of our system, which consists of two main parts of schema matching and query processing, in part A and part B respectively. The components of schema matching which operate during the set-up time of the system are as follows:

- Attribute similarity computing: this component computes the attribute name similarity between every two source attributes.

– FD derivation: this component derives functional dependencies from data, which is an optional component of the system and is only used in the cases where functional dependencies are not given to the system.
– Distance assignment: this component uses attribute pairs similarity and functional dependencies for generating the distance function.
– Schema matching: this component uses the distance function for generating a set of probabilistic mediated schemas.
– Single schema building: this component generates one mediated schema for the user by using the generated probabilistic mediated schemas.

Part B of Figure 1 depicts the components of the query processing part. We include these components in the architecture of our system to provide a complete picture of a data integration system but our focus is on the schema matching part (part A). The components of part B which operate at query evaluation time are as follows:

– Query reformulation: This component uses the probabilistic mediated schemas to reformulate the user query posed against the mediated schema to a set of queries posed over the data sources.
– Query result aggregation: This component combines the results of reformulated queries and assigns a probability to every tuple in the result, based on both the probabilities of the mediated schemas and the dependency among data sources.

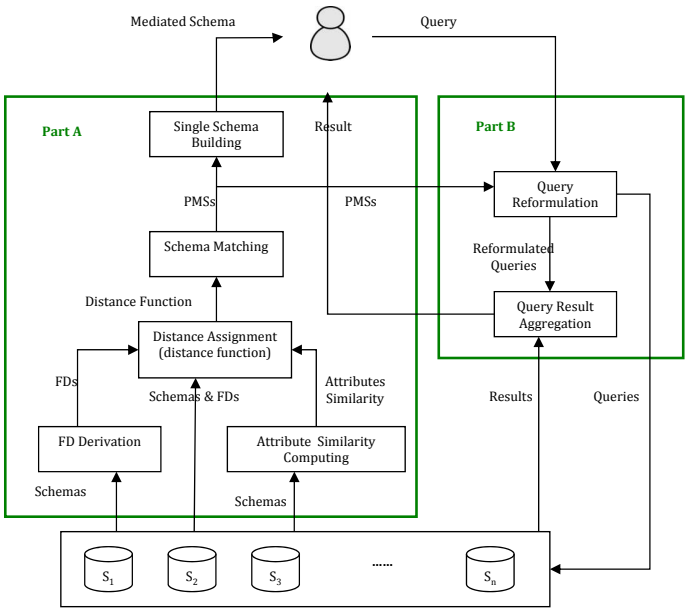

Fig. 1. Architecture of our data integration system

3.2 Mediated Schema Generation

To build the mediated schema automatically, we cluster the source attributes by putting semantically equivalent attributes in the same cluster. In order to do this without using domain expert's knowledge, we rely on two clues: attributes' name similarity and FDs between attributes. In some cases, attributes with the same semantics also have similar names, so attribute similarity alone can lead to good attribute clustering. However, this is not suitable for our underlying applications where there are many attributes with the same semantics which have completely different names, and vice versa. Therefore, as the second clue, we use FDs for identifying semantically equivalent attributes.

Typically, since we are not sure about the semantics of attributes, uncertainty arises while clustering the attributes. To capture the uncertainty and improve the result of clustering over time, we use a probabilistic data model with which we create probabilistic mediated schemas (instead of only one mediated schema). This helps us to better capture the inherent uncertainty in clustering the attributes. In traditional data integration systems, queries are posed over one mediated schema, but in a probabilistic approach we generate several mediated schemas.

Distance Assignment To cluster the source attributes, we use a clustering algorithm that works based on a distance matrix (i.e. the distance between every two attributes). Specifically we use the *single-link CAHC* (Constrained Agglomerative Hierarchical Clustering) algorithm [12]. The problem is how to assign the distances between the attributes in order to obtain good mediated schemas. In our work, we do this by using the attributes' name similarity as well as some heuristics we introduce about FDs.

Distance is a number between 0 and 1. We set the distance between attributes from the same source to a value equal to 1 (i.e. maximum value), because it is not reasonable to put such attributes in the same cluster. For other attribute pairs, if the FD heuristics are not applicable to them, we set the distance equal to 1 minus the name similarity (which is also a number between 0 and 1), e.g. if a pair's name similarity is 0.3, then we set the distance to 0.7. For the other cases, we use our heuristics which will be described later.

Before describing our heuristics, let us first define *Match* and *Unmatch* concepts. Consider a_1 and a_2 as two typical attributes. If we want to increase their chance of being put in the same cluster, we set their distance to *MD* (i.e. Match Distance) which is 0 or a number very close to 0. In this case, we say that we matched a_1 with a_2, and we show this by $Match(a_1, a_2)$. In contrast, if we want to decrease their chance of being put in the same cluster, then we set their distance to *UMD* (i.e. Un-Match Distance) which is 1 or a number very close to 1. In this case, we say that we *unmatched* a_1 and a_2 and we show this by $Unmatch(a_1, a_2)$.

FD Heuristics We use a number of heuristic rules related to FDs in order to assign the distance of attributes. Let us use the following example to illustrate our heuristics.

Example 3. Consider two source schemas, both describing a university course schedule. In this example, primary keys are underlined; F_1 and F_2 are the sets of FDs of S_1 and S_2 respectively:
$S_1(\underline{term, c\#, section\#}, coursename, instructor, name, time, room)$
$S_2(\underline{semester, course, sec\#}, name, instructor, ins\_name, location)$
$F_1 = \{c\# \rightarrow coursename, instructor \rightarrow name\}$
$F_2 = \{course \rightarrow name, instructor \rightarrow ins\_name\}$

Heuristic 1 *Let S_p and $S_q, p \neq q$, be two source schemas. Then,*

$$Match(a_{p,i}, a_{q,k}) \Rightarrow unmatch(a_{p,i}, a_{q,l}) \wedge unmatch(a_{q,k}, a_{p,j})$$

where $a_{p,i} \in att(S_p), a_{p,j} \in att(S_p) \setminus \{a_{p,i}\}, a_{q,k} \in att(S_q), a_{q,l} \in att(S_q) \setminus \{a_{q,k}\}$.

The reason behind heuristic 1 is that each attribute can be matched with at most one attribute of the other source.

Heuristic 2 *Let $fd_p : a_{p,i} \rightarrow a_{p,j}$ and $fd_q : a_{q,k} \rightarrow a_{q,l}$ be two FDs, where $fd_p \in F_p, fd_q \in F_q, p \neq q$. Then, $similarity(a_{p,i}, a_{q,k}) > t_L \Rightarrow Match(a_{p,j}, a_{q,l})$ where t_L is a certain threshold and similarity is a given similarity function.*

The reason behind heuristic 2 is that we consider the set of facts that the two sources are assumed to be from the same domain, and both attributes $a_{p,j}$ and $a_{q,l}$ are functionally determined by the attributes $a_{p,i}$, and $a_{q,k}$ respectively, which themselves have close name similarity. Thus, we heuristically agree that: the probability of $Match(a_{p,j}, a_{q,l})$ is higher than that of $Match(a_{p,j}, a_{q,s})$ and $Match(a_{q,l}, a_{p,r})$, where $a_{q,s} \in att(S_q) \setminus \{a_{q,l}\}$ and $a_{p,r} \in S_p \setminus \{a_{p,j}\}$. Therefore, in such a case we match $a_{p,j}$ with $a_{q,l}$ to reflect this fact. Note that this heuristic has a general form in which there are more than one attribute on the sides of the FDs (see Section 3.2).

By applying heuristic 2 on Example 3, we have the FD $instructor \rightarrow name$ from S_1, and $instructor \rightarrow ins\_name$ from S_2. There is only one attribute at the left side of these FDs, and their name similarity is equal to 1 that is the maximum similarity value. Thus, we match the *name* with the *ins_name* which appear on the right side of these FDs. Notice that in this example, FDs guided us to recognize that the *name* in S_2 is in fact the instructor's name, and not the course's name. This kind of mistake is typically made by approaches which only rely on name similarity for attribute matching.

Heuristic 3 *Let PK_p and $PK_q, p \neq q$, be the primary keys of S_p and S_q respectively. Then,*

$$(\exists a_{p,i} \in PK_p, a_{q,j} \in PK_q \mid (a_{p,i}, a_{q,j}) = \underset{a_p \in PK_p, a_q \in PK_q}{\arg\max} similarity(a_p, a_q)) \wedge$$
$$(similarity(a_{p,i}, a_{q,j}) > t_{PK}) \Rightarrow Match(a_{p,i}, a_{q,j})$$

where t_{PK} is a certain threshold and similarity is a given similarity function.

The reason behind heuristic 3 is simple. Since we assume sources are from the same domain, there are a number of specific attributes which can be part of the primary key. Although these attributes may have different names in different sources, it is reasonable to expect that some of these attributes from different sources can be matched together. Obviously, we can set t_{PK} to a value less than the value we set for t_L because typically the probability of finding matching attributes in the primary key attributes is higher than the other attributes. After matching $a_{p,i}$ with $a_{q,j}$, we remove them from PK_p and PK_q respectively, and continue this process until the similarity of the pair with the maximum similarity is less than the threshold t_{PK} or one of the PK_p or PK_q has no more attributes to match.

Now we apply heuristic 3 to Example 3. It is reasonable to match the attributes: *term*, *c#*, and *section#* of S_1 with *semester*, *course*, and *sec#* of S_2 rather than with other attributes of S_2, and vice versa. The attribute pair with the maximum similarity is (*section#*, *sec#*). If we choose a good threshold, we can match these attributes together. The similarity of other attribute pairs is not high enough to pass the wisely selected threshold values.

Heuristic 4 *Let PK_p and $PK_q, p \neq q$, be the primary keys of S_p and S_q respectively. Then,*

$$(\exists a_{p,i} \in PK_p, a_{q,j} \in PK_q, fd_p \in F_p, fd_q \in F_q \mid fd_p : a_{p,i} \rightarrow R_p, fd_q : a_{q,j} \rightarrow R_q) \Rightarrow Match(a_{p,i}, a_{q,j}) \quad (1)$$

and also

$$(RHS(1) \wedge R_p = \{a_{p,r}\} \wedge R_q = \{a_{q,s}\}) \Rightarrow Match(a_{p,r}, a_{q,s}) \quad (2)$$

We can apply heuristic 4 when we have two attributes in two primary keys which each of them is the single attribute appearing at the left side of a FD. In this case, we match these attributes with each other(rule 1). We also match the attributes on the right sides of the two FDs if there is only one attribute appearing at the right side of them (rule 2).

By applying heuristic 4 on Example 3, we match *c#* with *course* which is a right decision. We do this because of the two FDs: *c#* $\rightarrow$ *coursename* and *course* $\rightarrow$ *name*. We also match *coursename* with *name* which are the only attributes appearing at the right side of these FDs. Had we used name similarity only, we would have very likely matched *coursename* with *course* for example, which is a wrong decision.

Heuristic 5 *Let PK_p and $PK_q, p \neq q$, be the primary keys of S_p and S_q respectively. Then,*

$$(\forall a_{p,r} \in PK_p \setminus \{a_{p,i}\}, \exists a_{q,s} \in PK_q \setminus \{a_{q,j}\} \mid Match(a_{p,r}, a_{q,s})) \wedge (|PK_p| = |PK_q|) \Rightarrow Math(a_{p,i}, a_{q,j})$$

We can apply heuristic 5 when all attributes of PK_p and PK_q have been matched, and only one attribute is left in each of them. We match these two attributes with each other hoping that they are semantically the same. Coming back to Example 3, there is only one attribute left in each of the primary keys that we have not yet matched (i.e. *term*, *semester*) that we can match using this heuristic.

Distance Assignment Algorithm Algorithm 1 describes how we assign distances to attribute pairs and build the distance matrix that is used in schema matching. It takes as input the source schemas, the set of PK (Primary Key) of every source, and the set of FDs of every source such that each $fd \in F_i$ is of the form $L \rightarrow R$, where L and R are attribute subsets from source S_i, i.e. $L, R \subseteq att(S_i)$. Notice that the FD related to the PK of S_i is removed from F_i. Steps 2-10 of the algorithm find FD pairs from different sources which their left sides match together and then try to match attribute pairs on the right sides of these FDs. Steps 5-7 find the attribute pairs (a_p, a_q) whose similarity is maximum. If the similarity of a_p and a_q is more than threshold t_R, their distance is set to MD (Match Distance), and the distances between each of them and any other source-mates are set to UMD (Unmatch Distance). The algorithm uses the $DoMatch$ procedure for matching and unmatching attributes. It gets the attributes which should be matched as parameter, matches them, and unmatches every one of them with the other ones' source-mates. Generally, whenever the algorithm matches two attributes with each other, it also unmatches the two of them with the other one's source-mates because every attribute of a source can be matched with at most one attribute of every other source. Steps 8-10 remove the matched attributes from the list of unmatched attributes, and repeats the matching process if there are still some attributes remaining for matching.

Step 3 uses the $IsMatch$ function. This function takes as parameter the left sides of two FDs and returns true if they can be matched together, otherwise it returns false. It first checks whether the input parameters are two sets of the same size. Then, it finds the attribute pair with maximum name similarity and treats it as matched pair by removing the attributes from the list of unmatched attributes if their similarity is more than threshold t_L. It repeats the matching process until there is no more attribute eligible for matching. After the matching loop is over, the function returns true if all attribute pairs have been matched together, otherwise it returns false which means the matching process has not been successful.

Notice that we do not reflect the matching of attributes of the left sides of FDs in the distance matrix. The reason is that for these attributes (in contrast to those on the right side), the matching is done just based on attribute name similarity and not the knowledge in FDs.

In these algorithms, we use three different similarity thresholds (i.e. t_L, t_R, and t_{PK}). We do this to have more flexibility in the matching but we need to set them carefully. If we set them to high values, we prevent wrong matching but may miss some pairs that should have been matched. On the other hand, if we set thresholds to low values, we increase the number of correctly matched pairs

but also increase the number of wrongly matched pairs. In other words, setting the threshold values is a trade off between precision and recall. Aside from this, the inequality between them is important as we explain below. We know that t_L is the similarity threshold for matching attributes at the left sides of FDs. Since the matching of left sides of FDs is taken as evidence for matching the right sides of them, t_L needs to be chosen carefully. Setting it to low values, results in wrong matchings. On the other hand, we use t_R as similarity threshold for matching attributes on the right sides of FDs. Since we already have evidence for matching them, we can be more relaxed in setting t_R by setting it to values lower than t_L. The same argument goes for the value of t_{PK}. t_{PK} is the similarity threshold for matching PK attributes. Since these attributes are a subset of source attributes, it is reasonable to set t_{PK} to lower values than t_L and t_R.

Coming back to Algorithm 1, steps 11-26 apply PK heuristics to every PK pair and try to match their attributes based on these heuristics. Steps 13-18 check every attribute pair of two PKs to see if they are the only attributes at the left sides of two FDs. If yes, then these attributes are matched together. Steps 19-24 find the attribute pair with the maximum name similarity and if it is more than threshold t_{PK}, the attributes are matched together. The matching process continues until there is at least one attribute in every PK and the similarity of the attribute pair with the maximum similarity is more than threshold t_{PK}. After the matching process, if each of the two PKs has only one attribute left, their attributes are matched with each other by steps 25-26.

Steps 27-31 set the distances of attribute pairs which have not been computed by the heuristic rules. Step 28 checks if the attributes are from the same source, in which case their distance is set to UMD; otherwise the distance is set to their name similarity by step 31.

Steps 32-33 perform a transitive closure over the match and unmatch constraints using the following two rules:

1. If $match(a_i, a_k)$ and $unmatch(a_k, a_j)$, then $unmatch(a_i, a_j)$. If a_i and a_k are in the same cluster and a_k and a_j are not in the same cluster, then it is clear that a_i and a_j should not be in the same cluster either.
2. If $match(a_i, a_k)$ and $match(a_k, a_j)$, then $match(a_i, a_j)$. If a_i and a_k are in the same cluster and a_k and a_j are also in the same cluster, then it is obvious that a_i and a_j should also be in the same cluster.

Step 34 deals with the symmetric property of the distance between attributes. It ensures that the returned distance is independent from the order of attributes.

The matching and unmatching decisions made by a distance function should be consistent with each other. More precisely, a consistent distance function should not satisfy the following condition:

$$\exists a_i, a_j, a_k \in A \mid match(a_i, a_j) \wedge match(a_j, a_k) \wedge unmatch(a_i, a_k). \quad (3)$$

The following proposition shows that our distance function is consistent.

Proposition 1. *Algorithm 1 returns a consistent distance function.*

Proof. We first show that if inconsistency exists, it is removed by step 32 of the algorithm, i.e. the first transitive closure rule. Then, we show that order of applying the transitive closure rules in Algorithm 1 is the only correct order. Let us prove the first part. Suppose steps 1-31 of the algorithm create an inconsistency so that condition (3) satisfies. Then, as the result of step 32 of the algorithm, either $match(a_i, a_j)$ changes to $unmatch(a_i, a_j)$ or $match(a_j, a_k)$ changes to $unmatch(a_j, a_k)$. It is clear that the inconsistency between a_i, a_j, and a_k is removed with either of the changes. Without the loss of generality, we assume that $match(a_i, a_j)$ changes to $unmatch(a_i, a_j)$. Then, if there exists $a_l \in A$, so that condition (3) satisfies for a_i, a_j, and a_l as a result of the change, step 32 removes it too. Thus, step 32 removes all of the inconsistencies in the cost of losing possibly correct match pairs.

Let us prove the second part. Suppose that steps 1-31 of the algorithm create an inconsistency so that condition (3) satisfies and we change the order of transitive closure rules. By first applying rule 2, $unmatch(a_i, a_k)$ changes to $match(a_i, a_k)$. However, we already unmatched a_i with a_k as the result of matching a_i with one of the source-mates of a_k, say a_l. Thus, we have: $match(a_k, a_i)$ and $match(a_i, a_l)$, which results in $match(a_k, a_l)$ by applying rule 2 to them. This means that we matched two attributes a_k and a_l from the same source. Thus, changing the order of transitive closure rules does not remove the inconsistency but propagates it.

Generating Clusters The distances between attributes are used for computing the distance between clusters in the clustering method, i.e. CAHC. By definition, the distance between two clusters is the minimum distance between the attributes from different clusters. The generation of attribute clusters proceeds as follows. In the first step, we begin with a number of clusters each having only one attribute, i.e. n clusters, when n is the number of attributes. Then, we find the two clusters with the minimum distance, and we combine them. We continue this process until the minimum distance between clusters is greater than or equal to the value we have set for the distance between attributes from the same source. We continue merging clusters until the number of clusters becomes equal to the arity of the source (i.e. the number of attributes in that source) with the maximum arity.

Since each mediated schema is a clustering of attributes, we have in fact created one mediated schema at every step. However, these mediated schemas are not all of the same quality. We would like to select the best one(s). We defer the definition of the quality of a mediated schema to Section 4 where we formally define the quality metrics for measuring the quality of a mediated schema. Since for all generated mediated schemas we do not let unmatched attributes to be put in the same cluster, we count the number of matched pairs which has been respected by the mediated schema, as a metric for ranking mediated schemas. We call this metric the FD-point. As we will show in Section 4, there is a correlation between the FD-point and the quality of the mediated schema. Thus, we select the mediated schema(s) with the maximum FD-point, as the schema(s) with the

Algorithm 1 Distance Assignment

Input: 1) Source schemas $S_1, \ldots, S_n$; 2) The sets of FDs $F_1, \ldots, F_n$ (the FDs related to PK are omitted); 3) $P = \{PK_1, \ldots, PK_n\}$ The set of primary keys of all sources.

Output: Distance matrix $D[m][m]$.

compute $A = \{a_1, \ldots, a_m\}$ the set of all source attributes
// match attributes on the right sides of FDs
for all FD pair $fd_i \in F_k, fd_j \in F_l, k \neq l$ **do**
  if $IsMatch(L_i, L_j)$ **then**
    make local copies of fd_i, fd_j
    find the attribute pair $a_p \in R_i, a_q \in R_j$ with the maximum similarity s
    if $s > t_R$ **then**
      $DoMatch(a_p, a_q)$
      $R_i \leftarrow R_i \setminus \{a_p\}$; $R_j \leftarrow R_j \setminus \{a_q\}$
      if $\mid R_i \mid > 0$ and $\mid R_j \mid > 0$ **then**
        go to 5
// match PK attributes
for all pair $PK_i, PK_j \in P$, where they are PKs of S_i and S_j respectively **do**
  make local copies of PK_i and PK_j
  for all pair $a_p \in PK_i, aq \in PK_j$ **do**
    if $\exists fd_k \in F_i$ and $fd_l \in F_j$ such that $L_k = \{a_p\}$ and $L_l = \{a_q\}$ **then**
      $DoMatch(a_p, a_q)$
      $PK_i \leftarrow PK_i \setminus \{a_p\}$; $PK_j \leftarrow PK_j \setminus \{a_q\}$
      if $R_k = \{a_s\}$ and $R_l = \{a_t\}$ **then**
        $DoMatch(a_p, a_q)$
  find the attribute pair $a_p \in PK_i$ and $a_q \in PK_j$ with maximum similarity s
  if $s > t_{PK}$ **then**
    $DoMatch(a_p, a_q)$
    $PK_i = PK_i \setminus \{a_p\}$; $PK_j = PK_j \setminus \{a_q\}$
    if $\mid PK_i \mid > 0$ and $\mid PK_j \mid > 0$ **then**
      go to 19
  if $PK_i = \{a_p\}$ and $PK_j = \{a_q\}$ **then**
    $DoMatch(a_p, a_q)$
for all attribute pair $a_i, a_j \in A$ which $D[a_i][a_j]$ has not been computed yet **do**
  if $a_i, a_j \in S_k$ (the same source) **then**
    $D[a_i][a_j] \leftarrow UMD$
  else
    $D[a_i][a_j] \leftarrow similarity(a_i, a_j)$
$\forall a_i, a_j, a_k \in A$ **if** $(D[a_i][a_k] = MD$ and $D[a_k][a_j] = UMD)$ **then** $D[a_i][a_j] \leftarrow UMD$
$\forall a_i, a_j, a_k \in A$ **if** $(D[a_i][a_k] = MD$ and $D[a_k][a_j] = MD)$ **then** $D[a_i][a_j] \leftarrow MD$
$\forall a_i, a_j \in AD[a_i][a_j] \leftarrow D[a_j][a_i]$

highest quality. We assign equal probabilities to all selected mediated schemas, since we do not have any other metrics for differentiating between the selected mediated schemas.

Algorithm 2 Schema Matching

Input: 1) Source schemas $S_1, \ldots, S_n$; 2) Distance matrix $D[m][m]$; 3) Number of needed mediated schemas k.

Output: A set of probabilistic mediated schemas.

1: compute $A = \{a_1, \ldots, a_m\}$ the set of all source attributes
2: let C be the set of clusters c_i such that $c_i = \{a_i\}, a_i \in A, i \in [1, m]$
3: $M \leftarrow C$
4: find two clusters $c_i, c_j \in C$ having the minimum distance d_{min} while distance d_{ij} between c_i and c_j is computed as follows:
5: **if** $\exists a_k \in c_i, a_l \in c_j, a_k, a_l \in S_p$ **then**
6: $d_{ij} \leftarrow \infty$
7: **else**
8: $d_{ij} \leftarrow Min(D[a_k][a_l]), a_k \in c_i, a_l \in c_j$
9: **if** $d_{min} \neq \infty$ **then**
10: merge c_i with c_j
11: Add the newly added mediated schema to M
12: go to 4
13: **for each** $C_i \in M$ compute the $FDpoint_i$ as the number of attribute pairs recommended by distance matrix and respected by C_i
14: $FDpoint_{max} \leftarrow Max(FDpoint_i), C_i \in M$
15: $M \leftarrow \{C_i \mid C_i \in M, FDpoint_i = FDpoint_{max}\}$
16: **if** $k <\mid M \mid$ **then**
17: select k mediated schemas randomly from M
18: assign probability $\frac{1}{k}$ to every selected mediated schema and return them
19: **else**
20: assign probability $\frac{1}{M}$ to every $C_i \in M$ and return them

Schema Matching Algorithm Algorithm 2 describes how we match different source schemas and create probabilistic mediated schemas. This algorithm takes as input the source schemas, distance matrix, and the needed number of mediated schemas which is specified by the user. Steps 1-2 create the first mediated schema by putting every attribute in a cluster. The algorithm stores all created mediated schemas in the set M, and so does for the first created mediated schema in step 3.

Steps 4-8 look for the two clusters with the minimum distance while the distance between two clusters is defined as follows: if the clusters have two attributes from the same source, the distance between them is infinity; otherwise the minimum distance between two attributes, each from one of the two clusters, is regarded as the distance between the two clusters. Steps 9-12 merge these clusters together and store this newly created mediated schema in M and continues this process by going to step 4. The necessary condition for merging clusters is that their distance should not be equal to infinity. We get the infinity as the minimum distance between clusters when every two clusters have attributes from the same source. In such a case, we stop creating the mediated schemas.

For every created mediated schema, Step 13 computes its FD-point, which is a metric for measuring the quality of mediated schemas and for selecting only the high quality ones. Distance matrix recommends some attribute pairs to be

put in the same cluster by returning their distance as MD. FD-point is defined as the number of these recommendations which are respected by the mediated schema. Steps 14-15 select the mediated schemas with the maximum FD-point. We call them as eligible mediated schemas.

If the user specifies the number of mediated schemas (k) which she needs, steps 16-20 return k randomly selected mediated schemas to the user. Since the algorithm has no means for differentiating between eligible mediated schemas, it assigns equal probabilities to all returned mediated schemas.

On-the-fly Adding of Data Sources IFD starts with a given set of sources and ends up generating several mediated schemas from these sources. A useful property of IFD is that it allows new sources to be added to the system on the fly. Let S_{n+1} be the source which we want to add. By comparing S_{n+1} with each $S_i, i \in [1..n]$, we can compute the distance between every attribute of S_{n+1} and every attribute of S_i in the same way that we did for computing the distances between the attributes of $S_1..S_n$. After computing the distances, we consider every PMS, say $M_j, j \in [1..k]$ and for every attribute $a_p \in S_{n+1}$, we find the closest attribute $a_q \in A$ and put a_p in the same cluster as that of a_q. We repeat this process for every PMS.

This is a useful property of IFD which is needed in the contexts which we do not have all sources at hand when we start setting up the data integration application and we need to add them incrementally when they become available.

3.3 Schema Mapping

The result of schema matching is fed to the schema mapping algorithm for generating the mappings which are used in query processing. In our data integration solution, we assume that every attribute in the data sources can be matched with at most one attribute in other data sources which means we only consider one-to-one mappings. We do this for simplicity and also because this kind of mapping is more common in practice. We also assume single-table data sources. These two assumptions greatly simplify the schema mapping algorithm. For generating the mappings, we can rely on PMSs which implicitly contain the mappings for answering user queries. We do not provide the details of the schema mapping algorithm here due to its simplicity and also lack of space.

3.4 Cost Analysis

In this section, we study the execution costs of our schema matching and distance function algorithms.

Theorem 1. *Let m be the number of the attributes of all sources, then the running time of algorithms 1 and 2 together is $\theta(m^3)$.*

Proof. The basis for our schema matching algorithm is the single-link CAHC (Constrained Agglomerative Hierarchical Clustering) algorithm in which the number of clusters is determined by the arity of the source with the maximum

arity. Let m be the number of the attributes of all sources. The time complexity of the single-link AHC algorithm implemented using next-best-merge array (NBM) is $\Theta(m^2)$ [13].

Let us now analyze the running time of the distance function algorithm. Most of the algorithm is devoted to matching left and right sides of FDs, or the attributes of PKs. Let c be the arity of the source with the maximum arity, and f the maximum number of FDs that a source may have, which is a constant. The number of attributes on the left and right side of a FD is at most equal to the arity of its source, so its upper bound is c. Thus, matching both sides of two FDs takes $\Theta(c^2)$ time which is equal to $\Theta(1)$ because c is a constant. This argument also holds for matching PKs' attributes because the algorithm only checks the FDs of the two sources (which each one at most has f FDs), not the FDs of all sources.

Let n be the number of sources, then we have at most $f \times n$ FDs. The algorithm checks every FD pair for matching. Thus, it takes $\frac{f \times n \times (f \times n-1)}{2} \times \Theta(1)$ time for matching FDs which is equal to $(n^2 \times f^2)$. By taking f, i.e. the maximum number of FDs, as a constant, the complexity is $\Theta(n^2)$. In the same way, the time complexity for matching PKs is $\Theta(n^2)$.

The transitive closure part of the algorithm is done in $\theta(m^3)$ time, where m is the total number of attributes. The last part of the algorithm that guarantees symmetric property takes $\theta(m^3)$. Since the number of attributes is at least the number of sources, we have $m \geq n$. Thus, the transitive closure of attributes dominates all other parts of the algorithm and the running time of the algorithm is $\theta(m^3)$. As a result, the running time of the schema matching and the distance function algorithms together is $\theta(m^3)$.

4 Performance Evaluation

In this section, we study the effectiveness of our data integration solution. In particular, we show the effect of using functional dependencies on the quality of generated mediated schemas. We compare our solution with the one presented in [4] which is the closest to ours. To examine the contribution of using a probabilistic approach, we compare our approach with two traditional baseline solutions that do not use probabilistic techniques, i.e. they generate only one single deterministic mediated schema.

The rest of this section is organized as follows. We first describe our experimental setup. Then we compare the performance of our solution with the competing approaches.

4.1 Experimental Setup

We implemented our system (IFD) in Java. We took advantage of Weka 3-7-3 classes [14] for implementing the hierarchical clustering component. We used the SecondString tool[1] to compute the Jaro Winkler similarity [15] of attribute names in pair-wise attribute comparison. We conducted our experiments on a Windows XP machine with Intel core 2 GHz CPU and 2GB memory.

[1] Secondstring. `http://secondstring.sourceforge.net/`

In our experiments, we set the number of mediated schemas (denoted as n) to 1000, which is relatively high, in order to return all eligible mediated schemas. Our experiments showed similar results when we varied n considerably (e.g. n = 5). The default values for the parameters of our solution are as follows. We set similarity threshold for PK attributes (t_{PK}) to 0.7, similarity threshold for attributes on the left side of functional dependencies (t_L) to 0.9, similarity threshold for attributes on the right side of functional dependencies (t_R) to 0.8, the distance between attributes being matched (MD) to 0, and the distance between attributes being unmatched (UMD) to 1.

We evaluated our system using a dataset in the university domain. This dataset[2] consists of 17 single-table schemas which we designed ourselves. For having variety in attribute names, we used Google Search with "computer science" and "course schedule" keywords and picked up the first 17 related results. For every selected webpage, we designed a single-table schema which could be the data source of the course schedule information on that webpage and we used data labels as attribute names of the schema. Also, we created primary key and functional dependencies for every schema using our knowledge of the domain.

To evaluate the quality of generated mediated schemas, we tested them against the schema which we created manually. Since each mediated schema corresponds to a clustering of source attributes, we measured its quality by computing the precision, recall, and F-measure of the clustering. Let TP (True Positive) be the number of attribute pairs that are clustered correctly, FP (False Positive) be the number of attribute pairs that are clustered wrongly, and FN (False Negative) be the number of attribute pairs which are clustered together in the manually created schema but such pairs do not exist in the aforementioned schema. The three metrics are defined as follows: Precision: $P = \frac{TP}{TP+FP}$; Recall: $R = \frac{TP}{TP+FN}$; F-measure: $F = \frac{2 \times P \times R}{P+R}$.

We computed the metrics for each individual mediated schema, and summed the results weighted by their respective probabilities.

To the best of our knowledge, the most competing approach to ours (IFD) is that of Sarma et al. [4] which we denote by UDI as they did. Thus, we compare our solution with UDI as the most competing probabilistic approach. We implemented UDI in Java. UDI only considers attributes whose frequency are more than some threshold and omits the rest, but our approach considers all attributes regardless of their frequency. To be fair in comparison, we set the frequency threshold to 0 to consider all attributes. We used the same tool in our approach for computing pair-wise attribute similarity as in UDI. Also, we set the parameters edge-weight threshold and error bar to 0.85 and 0.02 respectively. Since the time complexity of UDI approach is exponential to the number of uncertain edges, we selected the above values carefully to let it run.

To examine the performance gain of using a probabilistic technique, we considered two baseline approaches that create a single mediated schema:

[2] The dataset is available at `http://www.science.uva.nl/CO-IM/papers/IFD/IFD-test-dataset.zip`

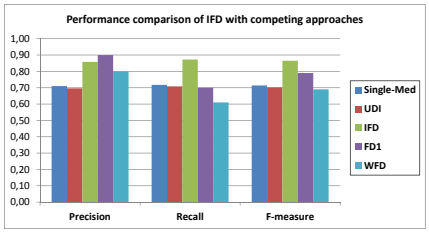

Fig. 2. Performance comparison of IFD with competing approaches

- FD1: creates a deterministic mediated schema as follows. In Algorithm 2, we count the number of FD recommendations and obtain the maximum possible FD-point, then we stop at the first schema which gets this maximum point.
- SingleMed: creates a deterministic mediated schema based on Algorithm 4.1 in [4]. We set frequency threshold to 0 and the edge weight threshold to 0.85.

Also, to evaluate the contribution of using functional dependencies in the quality of generated mediated schemas, we considered Algorithm 2 without taking advantage of the FD recommendations (WFD) and compared it to our approach.

4.2 Results

Quality of Mediated Schemas In this section, we compare the quality of mediated schemas generated by our approach (IFD) with the ones generated by UDI and other competing approaches.

Figure 2 compares the results measuring precision, recall, and F-measure of IFD, UDI, Single-Med, FD1, and WFD. It shows that IFD obtains better results than UDI. It improves precision by 23%, recall by 22%, and F-measure by 23%.

Figure 2 also shows the contribution of using FD recommendations in the quality of the results. WFD (Without FD) shows the results of our approach without using FD recommendations. It is obvious that using these recommendations has considerable effect on the results.

Furthermore, Figure 2 shows the performance gain of using a probabilistic approach rather than a single deterministic schema approach. FD1 applies all of the FD recommendations to obtain the mediated schema with the maximum FD-point, then stops and returns the resulted mediated schema. On the other hand, IFD does not stop after applying all FD recommendations but since there is no further FD recommendation, it starts merging clusters based on the similarity of their attribute pairs. This increases recall considerably, but reduces precision a little because some pairs are clustered wrongly. Overall, IFD improves F-measure by 8% compared to FD1. On the other hand, this Figure shows that UDI does not get such performance gain compared to Single-Med which creates a single deterministic schema. This happens because UDI cannot select the high quality schemas among the generated schemas.

Scalability To investigate the scalability of our approach, we measure the effect of the number of sources (n) on its execution time. By execution time, we mean

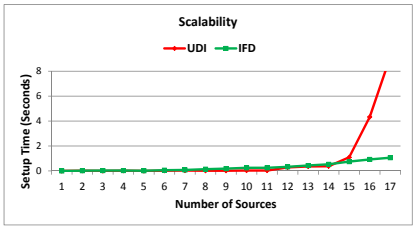

Fig. 3. Execution time of IFD and UDI (seconds)

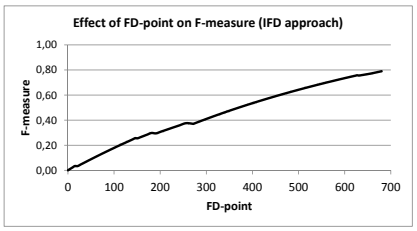

Fig. 4. Effect of FD-point on F-measure in IFD approach

the setup time needed to integrate n data sources. For IFD, the execution time equals to the execution time of computing distances using Algorithm 1 plus the execution time of generating mediated schemas using Algorithm 2. For UDI, we only consider the time needed to generate mediated schemas to be fair in our comparison. For UDI, the execution time is the time needed to create the mediated schemas.

Figure 3 shows how the execution times of IFD and UDI increase with increasing n up to 17 (the total number of sources in the tested dataset). The impact of the number of sources on the execution time of IFD is not as high as that of UDI. While in the beginning, the execution time of UDI is a little lower than IFD, it dramatically increases eventually. This is because the execution time of IFD is cubic to the number of the attributes of sources (see Section 3.4). But, the execution time of UDI is exponential to the number of uncertain edges. This shows that IFD is much more scalable than UDI.

Effect of FD-point In this section, we study the effect of FD-point on F-measure. Figure 4 shows how F-measure increases with increasing FD-point up to 680 which is the maximum possible value in the tested dataset. The starting point is when we have one cluster for every attribute. We have not used any recommendation at this point yet; as a result, $FD - point = 0$. Also it is clear that $precision = 1$ and $recall = 0$, thus $F - measure = 0$. As we begin merging clusters using recommendations, FD-point increases and this increases the F-measure as well. The increase in FD-point continues until it reaches its maximum possible value in the tested dataset. We consider all generated mediated schemas with maximum FD-point value as schemas eligible for being in the result set.

5 Related Work

There has been much work in the area of schema matching during the last three decades. Many of them have tried to provide automatic approaches to this problem (see [16] for a survey). They studied how to use various clues such as attribute names, data values, descriptions, data types, and constraints to identify the semantics of attributes and match them. An important class of approaches, which are referred to by constraint matchers, uses the constraints in schemas to determine the similarity of schema elements. Examples of such constraints are data types, value ranges, uniqueness, optionality, relationship types, and cardinalities.

OntoMatch [17], SemInt [18], SKAT [19], TranScm [20], DIKE [21], ARTEMIS [22], and CUPID [23] use this type of matcher. OntoMatch takes advantage of a constraint matcher which decides on matching the attributes based on being part of primary keys, foreign keys or being related to each other by some other constraint such as being of identical or compatible data types. ARTEMIS uses primary keys and foreign keys, and CUPID uses referential constraints for matching attributes. Our approach is different, since we use an uncertain approach for modelling and generating mediated schemas. Thus, the heuristic rules we use as well as the way we decrease the distance of the attributes is completely different. In addition, we take advantage of FDs. The proposals in [24] and [25] also consider the role of FDs in schema matching. However, our heuristic rules and the way we combine it with attribute similarity is completely different with these proposals. The Similarity Flooding algorithm (SF) [26] uses a similarity graph to propagate the similarity between attributes. Our work is different from SF in the ways that we do not propagate attribute similarity but instead we propagate the matching and unmatching of the attributes. In our work, we have not considered foreign keys, since we assumed single-table sources.

There has been a flurry of recent work on using probabilistic techniques for data integration [7, 27–29]. For instance in [7], Dong et al. introduce the concept of probabilistic schema matching. We used the concepts introduced in that paper as the foundation of uncertain data integration.

The closest work to ours is that of Sarma et al. [4] which we denote as UDI in this paper. UDI creates several mediated schemas with probabilities attached to them. To do so, it constructs a weighted graph of source attributes and distinguishes two types of edges: certain and uncertain. Then, a mediated schema is created for every subset of uncertain edges. Our approach has several advantages over UDI. The time complexity of UDI's algorithm for generating mediated schemas is exponential to the number of uncertain edges (i.e. attribute pairs) but that of our algorithm is PTIME (as shown in Section 3.4), therefore our approach is much more scalable. In addition, the quality of mediated schemas generated by our approach has shown to be considerably higher than that of UDI. Furthermore, the mediated schemas generated by our approach are consistent with all sources, while those of UDI may be inconsistent with some sources.

6 Conclusion

In this paper, we proposed IFD, a data integration system with the objective of automatically setting up a data integration application. IFD takes advantage of the background knowledge implied in FDs for finding attribute correlations and using it for matching the source schemas and generating the mediated schema. We built IFD on a probabilistic data model in order to model the uncertainty in data integration systems.

We validated the performance of IFD through implementation. We showed that using FDs can significantly improve the quality of schema matching (by 26%). We also showed the considerable contribution of using a probabilistic approach (10%). Furthermore, we showed that IFD outperforms UDI, its main

competitor, by 23% and has cubic scale up compared to UDI's exponential execution cost.

References

1. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 3rd Edition. Springer (2011)
2. Franklin, M.J., Halevy, A.Y., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Record **34**(4) (2005) 27–33
3. Madhavan, J., Cohen, S., Dong, X.L., Halevy, A.Y., Jeffery, S.R., Ko, D., Yu, C.: Web-scale data integration: You can afford to pay as you go. In: Proc. of CIDR. (2007)
4. Sarma, A.D., Dong, X., Halevy, A.Y.: Bootstrapping pay-as-you-go data integration systems. In: Proc. of SIGMOD. (2008)
5. Wang, D.Z., Dong, X.L., Sarma, A.D., Franklin, M.J., Halevy, A.Y.: Functional dependency generation and applications in pay-as-you-go data integration systems. In: Proc. of WebDB. (2009)
6. Akbarinia, R., Valduriez, P., Verger, G.: Efficient Evaluation of SUM Queries Over Probabilistic Data. TKDE **to appear** (2012)
7. Dong, X.L., Halevy, A.Y., Yu, C.: Data integration with uncertainty. VLDB J. **18**(2) (2009) 469–500
8. Fan, W., Geerts, F., Li, J., Xiong, M.: Discovering conditional functional dependencies. TKDE **23**(5) (2011) 683–698
9. Golab, L., Karloff, H.J., Korn, F., Srivastava, D., Yu, B.: On generating near-optimal tableaux for conditional functional dependencies. PVLDB **1**(1) (2008) 376–390
10. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: Tane: An efficient algorithm for discovering functional and approximate dependencies. Comput. J. **42**(2) (1999) 100–111
11. Ilyas, I.F., Markl, V., Haas, P.J., Brown, P.G., Aboulnaga, A.: Cords: Automatic generation of correlation statistics in db2. In: Proc. of VLDB. (2004)
12. Davidson, I., Ravi, S.S.: Using instance-level constraints in agglomerative hierarchical clustering: theoretical and empirical results. Data Min. Knowl. Discov. **18**(2) (2009) 257–282
13. Manning, C., Raghavan, P., Schutze, H.: Introduction to information retrieval. Volume 1. Cambridge University Press Cambridge (2008)
14. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explorations **11**(1) (2009) 10–18
15. Cohen, W.W., Ravikumar, P.D., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In: Proc. of IIWeb. (2003)
16. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4) (2001) 334–350
17. Bhattacharjee, A., Jamil, H.M.: Ontomatch: A monotonically improving schema matching system for autonomous data integration. In: Proc. of Conference on Information Reuse & Integration. (2009)
18. Li, W.S., Clifton, C.: Semint: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. Data Knowl. Eng. **33**(1) (2000) 49–84
19. Mitra, P., Wiederhold, G., Jannink, J.: Semi-automatic integration of knowledge sources. In: Proc. of Conference on Information Fusion. (1999)

20. Milo, T., Zohar, S.: Using schema matching to simplify heterogeneous data translation. In: Proc. of VLDB. (1998)
21. Palopoli, L., Terracina, G., Ursino, D.: Dike: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. Softw., Pract. Exper. **33**(9) (2003) 847–884
22. Castano, S., Antonellis, V.D., di Vimercati, S.D.C.: Global viewing of heterogeneous data sources. TKDE **13**(2) (2001) 277–297
23. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with cupid. In: Proc. of VLDB. (2001)
24. Biskup, J., Embley, D.W.: Extracting information from heterogeneous information sources using ontologically specified target views. Inf. Syst. **28**(3) (2003) 169–212
25. Larson, J.A., Navathe, S.B., Elmasri, R.: A theory of attribute equivalence in databases with application to schema integration. IEEE Trans. Software Eng. **15**(4) (1989) 449–463
26. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: Proc. of ICDE. (2002)
27. Florescu, D., Koller, D., Levy, A.Y.: Using probabilistic information in data integration. In: Proc. of VLDB. (1997)
28. Magnani, M., Montesi, D.: Uncertainty in data integration: current approaches and open problems. In: Proc. of Workshop on Management of Uncertain Data. (2007)
29. Magnani, M., Rizopoulos, N., McBrien, P., Montesi, D.: Schema integration based on uncertain semantic mappings. In: Proc. of Conference on Conceptual Modeling. (2005)