On entity resolution in probabilistic data
Ayat, S.N.

Citation for published version (APA):
Ayat, S. N. (2014). On entity resolution in probabilistic data

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 6
Pay-As-You-Go Data Integration Using Functional Dependencies

6.1 Introduction

In most of the applications that the entity resolution (ER) problem arises, the data resides in a number of data sources with heterogeneous schemas. Thus, before dealing with the ER problem, we first have to deal with the schema heterogeneity problem of data sources. Our aim in this chapter is to deal with this problem by setting up a data integration system without human intervention.

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 the mediated schema definition, schema matching and schema mapping. The mediated schema is the integrated 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. one source schema and the mediated schema in the popular Local As View (LAV) approach [100]. 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 [100, 130].

Setting up a full data integration system with a manually designed mediated schema requires significant human effort (e.g. by domain experts and database designers). On the other hand, the applications that we consider in this thesis need to start with a data integration application in a complete automatic setting for reducing human effort and development time, and putting more effort into improving it, as needed. This setting is referred to by pay-as-you-go data integration.

1The material of this chapter has been partially published in [19] and [18].
The goal of our research is to study how advanced of a starting point can we build a pay-as-you-go data integration system in a fully automated setting. Since probabilistic data models have shown to be promising [59, 115], we build our approach on a probabilistic data model to capture the uncertainty that arises during the schema matching process. Therefore, we generate a set of Probabilistic Mediated Schemas (PMSs). 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.

In database literature, the closest related work to ours is that of Sarma et al. [115] which is based on the PMSs proposed UDI (Uncertain Data Integration), as an uncertain data integration system. However, 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 [115].

6.1.1. Example. 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.

A high quality 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 both of the following schemas should be generated (we abbreviate \( hPhone \), \( oPhone \), \( hAddr \), \( oAddr \) as \( hP \), \( oP \), \( hA \), and \( oA \) respectively):

\[ M_1\{\{name, name\}, \{phone, hP\}, \{oP\}, \{address, hA\}, \{oA\}\} \]
\[ M_2\{\{name, name\}, \{phone, oP\}, \{hP\}, \{address, oA\}, \{hA\}\} \]

UDI does not consider attribute correlations. Thus, UDI may generate \( M_1 \) and \( M_2 \) together with many other schemas that do not always respect attribute correlations. As a results, by producing a large number of schemas which can easily be exponential, the desirable schemas get a very low probability.

Most attribute correlations are expressed through Functional Dependencies (FDs), which are defined among them. For example let \( F_1 \) and \( F_2 \) be the set of FDs of \( S_1 \) and \( S_2 \) respectively:

\[ F_1 = \{hPhone \rightarrow hAddr, oPhone \rightarrow oAddr\} \]
\[ F_2 = \{phone \rightarrow address\} \]

FDs in \( F_1 \) and \( F_2 \) show the correlation between the attributes in \( S_1 \) and \( S_2 \), respectively. For example, \( hPhone \rightarrow hAddr \) indicates that the two attributes \( hPhone \) and \( hAddr \) are correlated. Considering the pairs of FDs from different
6.2. Problem Definition

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

We assume that the functional dependencies between the attributes of sources are available. This is a reasonable assumption in the applications which we consider, because the data source providers are willing to provide the full database sources can help us with extracting these correlations and achieving the goal of generating mediated schemas that represent these correlations. For example, the FD pair: phone → address and hPhone → hAddr indicate that if we group phone and hPhone together, we should also group address and hAddr together, and similarly oPhone and oAddr.

In this chapter, we take advantage of the background knowledge which is implied within functional dependencies (FDs), for building a pay-as-you-go data integration system. The specific contributions of this chapter are the following.

- We propose IFD (Integration based on Functional Dependencies), a data integration system that takes into account attribute correlations by using functional dependencies, and captures uncertainty in mediated schemas using a probabilistic data model. Our system allows integrating a given set of data sources, as well as incrementally integrating additional sources, without needing to restart the process from scratch.

- 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 propose a new metric (i.e. FD-point) for ranking the generated mediated schemas in the clustering process, and selecting high quality ones.

- To validate our approach, we have implemented it as well as the 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 uncertain mediated schemas.

The rest of the chapter is organized as follows. In Section 6.2, we make our assumptions precise and define the problem. In Section 6.3, we briefly describe the architecture of our data integration system. In Section 6.4, we propose our approach for schema matching. We also analyze the execution cost of our proposed approach. Section 6.5 describes our performance validation. Section 6.6 discusses related work, and Section 6.7 concludes the chapter.

6.2 Problem Definition
design information, including functional dependencies. However, there are contexts, such as the web, for which functional dependencies are not available. For these applications, we can use one of the existing solutions, e.g. [77, 132], to derive functional dependencies from data. Second assumption, which we make for ease of presentation, is that the data model is relational.

Let us formally define some basic concepts, e.g. functional dependencies and mediated schemas, and then state the problem addressed in this chapter. 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 $\text{att}(S_i)$, and the set of all source attributes as $A$, i.e. $A = \cup_i \text{att}(S_i)$. For simplicity, we assume that each $S_i$ 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. $\text{att}(S_i)$, where each $fd_j, fd_j \in F_i$ is of the form $L_j \rightarrow R_j$ and $L_j \subseteq \text{att}(S_i), R_j \subseteq \text{att}(S_i)$. In every $F_i$, there is one fd of the form $L_p \rightarrow R_p$, where $R_p = \text{att}(S_i)$, i.e. $L_p$ is the primary key of $S_i$.

We assume one-to-one mappings of source attributes, meaning that each attribute can be matched with at most one attribute. 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 $M = \{A_1, \ldots, A_m\}$ as 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.

Let us formally define the concept of probabilistic mediated schemas (PMSs). The probabilistic mediated schemas (PMSs) 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 $\text{att}(S)$.
- $P(M_i) \in (0, 1]$.
- $\sum_{i=1}^{k} P(M_i) = 1$.

We use the precision, recall, and F-measure of the clustering for measuring the quality of the generated mediated schemas.

Now, we formally define 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 that have the highest F-measure.
6.3 System Architecture

The architecture of our data integration system, i.e. IFD, is shown in Figure 6.1. IFD consists of two main parts: schema matching (part A) and query processing (part B). 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.

The components of the query processing part is depicted in Part B of Figure 6.1. 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. Our simplifying assumptions, i.e. one-to-one mappings and single-table data sources, greatly simplify the way this component reformulates the user query.

• **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.

### 6.4 Schema Matching

In this section, we present the schema matching part of IFD. To match the schemas automatically, we cluster the source attributes by putting semantically equivalent attributes in the same cluster. We use a clustering algorithm that works based on a *distance function*, which determines the distance between every two attributes. Specifically, we use the single-link CAHC (Constrained Agglomerative Hierarchical Clustering) algorithm [53]. In the rest of this section, we first describe our distance function. Then, we describe our approach for schema matching. We then describe a useful feature of our approach. Finally, we analyze the execution cost of the proposed algorithms.

#### 6.4.1 Distance Function

Our schema matching algorithm uses a distance function for determining the distance between source attributes. To assign the distances between the attributes, we use the attributes' name similarity as well as some heuristics we introduce about FDs. In the rest of this section, we first describe our heuristics and then present the distance function algorithm.
6.4. Schema Matching

FD Heuristics

We use heuristic rules related to FDs in order to assign the distance of attributes. Before describing our heuristics, let us first define \textit{Match} and \textit{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 \(\text{Match}(a_1, a_2)\). 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 \(\text{Unmatch}(a_1, a_2)\).

Now, let us use the following example to illustrate the heuristics.

6.4.1. Example. Consider two source schemas, both describing bibliographical information of scientific papers. In this example, primary keys are underlined; \(F_1\) and \(F_2\) are the sets of FDs of \(S_1\) and \(S_2\) respectively:

\[
\begin{align*}
S_1 & \hspace{1em} (\text{author}, \text{title}, \text{year}, \text{institution}, \text{journal}, \text{issn}) \\
S_2 & \hspace{1em} (\text{name}, \text{paper_title}, \text{date}, \text{affiliation}, \text{journal}, \text{serial_no}, \text{volume}, \text{issue}) \\
F_1 & = \{ \text{author} \rightarrow \text{institution}, \text{journal} \rightarrow \text{issn} \} \\
F_2 & = \{ \text{name} \rightarrow \text{affiliation}, \text{journal} \rightarrow \text{serial_no} \}
\end{align*}
\]

6.4.2. Heuristic. Let \(S_p\) and \(S_q, p \neq q\), be two source schemas. Then,

\[
\text{Match}(a_{p,i}, a_{q,k}) \Rightarrow \text{unmatch}(a_{p,i}, a_{q,l}) \land \text{unmatch}(a_{q,k}, a_{p,j})
\]

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

Intuitively, this heuristic means that each attribute can be matched with at most one attribute of the other source.

6.4.3. Heuristic. 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, \(\text{similarity}(a_{p,i}, a_{q,k}) > t_L \Rightarrow \text{Match}(a_{p,j}, a_{q,l})\) where \(t_L\) is a certain threshold and \(\text{similarity}\) is a given similarity function.

In this heuristic, 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 \(\text{Match}(a_{p,j}, a_{q,l})\) is higher than that of \(\text{Match}(a_{p,j}, a_{q,s})\) and \(\text{Match}(a_{q,t}, a_{p,r})\), where \(a_{q,s} \in \text{att}(S_q\backslash\{a_{q,l}\})\) and \(a_{p,r} \in S_p\backslash\{a_{p,j}\}\). 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 6.4.1).

Let us apply heuristic 6.4.3 on Example 6.4.1. We have the FD \(\text{journal} \rightarrow \text{issn}\) from \(S_1\), and \(\text{journal} \rightarrow \text{serial_no}\) 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 issn with the serial no which appear on the right side of these FDs. Notice that approaches which only rely on name similarity, probably match issn with issue, which is a wrong decision.

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

$$\exists a_{p,i} \in PK_p, a_{q,j} \in PK_q \mid (a_{p,i}, a_{q,j}) = \arg \max_{a_p \in PK_p, a_q \in PK_q} \text{similarity}(a_p, a_q) \wedge$$

$$(\text{similarity}(a_{p,i}, a_{q,j}) > t_{PK} \Rightarrow \text{Match}(a_{p,i}, a_{q,j}))$$

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

Let us explain the idea behind heuristic 6.4.4. 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.

Coming back to Example 6.4.1, it is reasonable to match the attributes: author, title, and year of $S_1$ with name, paper title, and date of $S_2$ rather than with other attributes of $S_2$, and vice versa. The attribute pair with the maximum similarity is (title, paper title). 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.

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

$$\exists a_{p,i} \in PK_p, a_{q,j} \in PK_q, f_d_p \in F_p, f_d_q \in F_q \mid$$

$$f_d_p : a_{p,i} \rightarrow R_p, f_d_q : a_{q,j} \rightarrow R_q \Rightarrow \text{Match}(a_{p,i}, a_{q,j})$$

(6.1)

6.4.6. Heuristic. $(RHS(6.1) \wedge R_p = \{a_{p,r}\} \wedge R_q = \{a_{q,s}\}) \Rightarrow \text{Match}(a_{p,r}, a_{q,s})$

Heuristic 6.4.5 is applicable 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. 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 (heuristic 6.4.6).
Using heuristic 6.4.5 for Example 6.4.1, we match author with name which is a right decision. We do this because of the two FDs: author \( \rightarrow \) institution and name \( \rightarrow \) affiliation. We also match institution with affiliation which are the only attributes appearing at the right side of these FDs based on heuristic 6.4.6.

**6.4.7. HEURISTIC.** Let \( PK_p \) and \( PK_q, p \neq q, \) be the primary keys of \( S_p \) and \( S_q \) respectively. Then,

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

Heuristic 6.4.7 is applicable when all attributes of \( PK_p \) and \( PK_q \) have been matched, and only one attribute is left in each of them. In such case we match these two attributes with each other hoping that they are semantically the same.

Coming back to Example 6.4.1, there is only one attribute left in each of the primary keys that we have not yet matched (i.e. year, date) that we can match using this heuristic.

**Distance Function Algorithm**

Algorithm 8 describes how we combine the attributes’ name similarity and FD heuristics to build the distance function. Steps 2-12 of the algorithm apply heuristic 6.4.3. They look for 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. After finding such FDs, steps 5-10 repeatedly 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 9 remove the matched attributes from the list of unmatched attributes.

The \(IsMatch\) function, which is used by step 3, 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
Algorithm 8 Distance Function

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

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

1: compute $A = \{a_1, \ldots, a_m\}$ the set of all source attributes
2: for all FD pair $fd_i \in F_k, fd_j \in F_l, k \neq l$ do // heuristic 6.4.3
3: if $IsMatch(L_i, L_j)$ then
4: make local copies of $fd_i, fd_j$
5: repeat
6: find the attribute pair $a_p \in R_i, a_q \in R_j$ with the maximum similarity $s$
7: if $s > t_R$ then
8: $DoMatch(a_p, a_q)$
9: $R_i \leftarrow R_i \setminus \{a_p\}; R_j \leftarrow R_j \setminus \{a_q\}$
10: until $s > t_R$ and $|R_i| > 0$ and $|R_j| > 0$
11: for all pair $PK_i, PK_j \in P$, where they are PKs of $S_i$ and $S_j$ respectively do
12: make local copies of $PK_i$ and $PK_j$
13: for all pair $a_p \in PK_i, a_q \in PK_j$ do
14: 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
15: $DoMatch(a_p, a_q)$ // heuristic 6.4.5
16: $PK_i \leftarrow PK_i \setminus \{a_p\}; PK_j \leftarrow PK_j \setminus \{a_q\}$
17: if $\{R_k = \{a_p\}$ and $R_l = \{a_q\}$ then $DoMatch(a_p, a_q)$ // heuristic 6.4.6
18: repeat
19: find the attribute pair $a_p \in PK_i$ and $a_q \in PK_j$ with maximum similarity $s$
20: if $s > t_{PK}$ then
21: $DoMatch(a_p, a_q)$ // heuristic 6.4.4
22: $PK_i = PK_i \setminus \{a_p\}; PK_j = PK_j \setminus \{a_q\}$
23: until $s > t_{PK}$ and $|PK_i| > 0$ and $|PK_j| > 0$
24: if $\{PK_i = \{a_p\}$ and $PK_j = \{a_q\}$ then $DoMatch(a_p, a_q)$ // heuristic 6.4.7
25: for all attribute pair $a_i, a_j \in A$ which $D[a_i][a_j]$ has not been computed yet do
26: if $(a_i, a_j \in S_k)$ then $D[a_i][a_j] \leftarrow UMD$ // heuristic 6.4.2
27: else $D[a_i][a_j] \leftarrow similarity(a_i, a_j)$
28: for $a_i, a_j, a_k \in A$ if $(D[a_i][a_k] = MD$ and $D[a_k][a_j] = UMD$) then $D[a_i][a_j] \leftarrow UMD$
29: for $a_i, a_j, a_k \in A$ if $(D[a_i][a_k] = MD$ and $D[a_k][a_j] = MD$) then $D[a_i][a_j] \leftarrow MD$
30: for $a_i, a_j \in AD[a_i][a_j] \leftarrow D[a_j][a_i]$

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.

Please notice that we use three different similarity thresholds (i.e. $t_L, t_R$, and $t_{PK}$) to have more flexibility in the matching. However, 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 $t_R$.

In steps 11-24, we apply heuristics 6.4.4, 6.4.5, 6.4.6 and 6.4.7. Steps 13-17 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 18-23 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 $t_{PK}$. If each of the two PKs has only one attribute left, step 24 matches them together based on heuristic 6.4.7.

In steps 25-27, we set the distances of attribute pairs which have not been set yet. Step 26 applies heuristic6.4.2 by setting the distance of the attributes from the same source to $UMD$. The distance of other attribute pairs is set to their name similarity. Steps 28-29 perform a transitive closure over the match and unmatch constraints. Step 30 deals with the symmetric property of the distance function to ensure 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:

$$\exists a_i, a_j, a_k \in A \mid match(a_i, a_j) \land match(a_j, a_k) \land unmatch(a_i, a_k). \quad (6.2)$$

The following proposition shows that our distance function is consistent.

6.4.8. PROPOSITION. Algorithm 8 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 8 is the only correct order.
Let us prove the first part. Suppose steps 1-31 of the algorithm create an inconsistency so that condition (6.2) satisfies. Then, as the result of step 32 of the algorithm, either $\text{match}(a_i, a_j)$ changes to $\text{unmatch}(a_i, a_j)$ or $\text{match}(a_j, a_k)$ changes to $\text{unmatch}(a_j, a_k)$. 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 $\text{match}(a_i, a_j)$ changes to $\text{unmatch}(a_i, a_j)$. Then, if there exists $a_l \in A$, so that condition (6.2) 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 (6.2) satisfies and we change the order of transitive closure rules. By first applying rule 2, $\text{unmatch}(a_i, a_k)$ changes to $\text{match}(a_i, a_k)$. 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: $\text{match}(a_k, a_i)$ and $\text{match}(a_i, a_l)$, which results in $\text{match}(a_k, a_l)$ 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. □

### 6.4.2 Schema Matching Algorithm

We now describe the schema matching algorithm which works based on the CAHC clustering method. We use the distance function to compute the distance between clusters in the CAHC clustering method. Algorithm 9 describes how we create probabilistic mediated schemas. This algorithm takes as input the source schemas, distance matrix, and the needed number of mediated schemas ($k$) 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 4-11 repeatedly find 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. These clusters are merged together by step 9 if their distance is not equal to infinity, and the newly created mediated schema is added to $M$ by step 10. We consider 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.

To assign a rank to each generated mediated schema, we define FD-point as the number of matched pairs which has been respected by a mediated schema. For every created mediated schema, Step 12 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
Algorithm 9 Schema Matching

Input:
- Source schemas $S_1, \ldots, S_n$
- Distance matrix $D$ 
- 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: repeat
5: 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:
6:   if $(\exists a_k \in c_i, a_l \in c_j, a_k \in S_p)$ then $d_{ij} \leftarrow \infty$
7:   else $d_{ij} \leftarrow \min(D[a_k][a_l]), a_k \in c_i, a_l \in c_j$
8:   if $d_{min} \neq \infty$ then
9:     merge $c_i$ with $c_j$
10:    add the newly added mediated schema to $M$
11: until $d_{min} \neq \infty$
12: for each $C_i \in M$ compute the $FDpoint_i$ as the number of attribute pairs recommended by distance matrix and respected by $C_i$
13: $FDpoint_{max} \leftarrow \max(FDpoint_i), C_i \in M$
14: $M \leftarrow \{C_i | C_i \in M, FDpoint_i = FDpoint_{max}\}$
15: if $k < |M|$ then
16:    select $k$ mediated schemas randomly from $M$
17:    assign probability $\frac{1}{k}$ to every selected mediated schema and return them
18: else assign probability $\frac{1}{|M|}$ to every $C_i \in M$ and return them

6.4.3 Adding Data Sources Incrementally

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

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 13-14 select the mediated schemas with the maximum FD-point. We call them as eligible mediated schemas. Steps 15-18 return $k$ randomly selected eligible 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.

6.4.3 Adding Data Sources Incrementally

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.

### 6.4.4 Execution Cost Analysis

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

**6.4.9. Theorem.** Let $m$ be the number of the attributes of all sources, then the running time of Algorithm 8 and the schema matching algorithm 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 $\Theta(m^2)$ [90].

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 $\theta(m^3)$. □

6.5 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 [115] 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.

6.5.1 Experimental Setup

We implemented our system (IFD) in Java. We took advantage of Weka 3-7-3 classes [74] for implementing the hierarchical clustering component. We used the SecondString Java package\(^2\) to compute the Jaro Winkler similarity [139] 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.

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 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. This dataset, denoted by Courses, can be found in Appendix B.

\(^2\)http://secondstring.sourceforge.net
To evaluate the quality of generated mediated schemas, we tested them against the mediated 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. 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. [115] 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. 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:

- **FD1:** creates a deterministic mediated schema as follows. In Algorithm 9, 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 [115]. 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 9 without taking advantage of the FD recommendations (WFD) and compared it to our approach.

### 6.5.2 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 6.2 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 6.2 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 6.2 shows the performance gain of using a probabilistic approach rather than a single deterministic schema approach. FD1 applies all of
6.5. Performance Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleMed</td>
<td>0.85</td>
<td>0.92</td>
<td>0.88</td>
</tr>
<tr>
<td>UDI</td>
<td>0.80</td>
<td>0.87</td>
<td>0.83</td>
</tr>
<tr>
<td>IFD</td>
<td>0.75</td>
<td>0.90</td>
<td>0.82</td>
</tr>
<tr>
<td>FD1</td>
<td>0.70</td>
<td>0.85</td>
<td>0.77</td>
</tr>
<tr>
<td>WFD</td>
<td>0.65</td>
<td>0.80</td>
<td>0.73</td>
</tr>
</tbody>
</table>

Figure 6.2: Performance comparison of IFD with competing approaches

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 the setup time needed to integrate n data sources. For IFD, the execution time equals to the execution time of computing distances using Algorithm 8 plus the execution time of generating mediated schemas using Algorithm 9. 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 6.3 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 6.4.4), but
Chapter 6. Data Integration Using FDs

Figure 6.3: Execution time comparison of IFD and UDI

![Graph showing execution time comparison of IFD and UDI](image)

Effect of FD-point

In this section, we study the effect of FD-point on F-measure. Figure 6.4 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…

![Graph showing effect of FD-point on F-measure](image)

Figure 6.4: Effect of FD-point on F-measure in IFD approach

that 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 6.4 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.

6.6 Analysis against related Work

There has been much work in the area of automatic schema matching during the last three decades (see [108] for a survey). They studied how to use various clues 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. For instance, OntoMatch [26] and DIKE [101] use this type of matcher. Our approach is different, since we use an uncertain approach for modeling and generating mediated schemas. Thus, the heuristic rules that 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 [32] and [84] 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 than these proposals.

The closest work to ours is that of Sarma et al. [115] which we denoted as UDI in this chapter. 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 6.4.4), therefore our approach is much better 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.

6.7 Conclusion

In this chapter, we proposed IFD, a data integration system with the objective of automatically setting up a data integration application. We established an
advanced starting point for pay-as-you-go data integration systems. 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 (causing an improve by 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.