INFERRING INFORMATION ABOUT CORRESPONDENCES BETWEEN DATA SOURCES FOR DATASPACES

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2011

By
Chenjuan Guo
School of Computer Science

Contents

Word Count: 79,803

List of Tables

List of Figures

Abstract

Traditional data integration offers high quality services for managing and querying interrelated but heterogeneous data sources but at a high cost. This is because a significant amount of manual effort is required to help specify precise relationships between the data sources in order to set up a data integration system. The recent proposed vision of *dataspaces* aims to reduce the upfront effort required to set up the system. A possible solution to approaching this aim is to infer schematic correspondences between the data sources, thus enabling the development of automated means for bootstrapping dataspaces.

In this thesis, we discuss a two-step research programme to automatically infer schematic correspondences between data sources. In the first step, we investigate the effectiveness of existing schema matching approaches for inferring schematic correspondences and contribute a benchmark, called MatchBench, to achieve this aim. In the second step, we contribute an evolutionary search method to identify the set of entity-level relationships (*ELRs*) between data sources that qualify as entity-level schematic correspondences. Specifically, we model the requirements using a vector space model. For each resulting *ELR* we further identify a set of attribute-level relationships (*ALRs*) that qualify as attribute-level schematic correspondences. We demonstrate the effectiveness of the contributed inference technique using both MatchBench scenarios and real world scenarios.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisors, Prof. Norman W. Paton and Dr. Alvaro A. A. Fernandes, for their insightful supervision and guidance. I could not have imagined having better supervisors for my Ph.D study. I would also like to thank my advisor, Prof Andrei Voronkov, for his comments on my research. I would like to extend my appreciation to Dr. Cornelia Hedeler and Dr. Khalid Belhajjame for their helpful suggestions. Thanks also to School of Computer Science in the University of Manchester for providing a brilliant Ph.D. training and research environment.

I would also like to thank Dr. Kwanchai Eurviriyanukul who has been so kind and willing to take some time to help me in the beginning of my Ph.D. study. Thanks to Lily S. Safie, Lu Mao and many of my CS friends for their continuous encouragement and companion.

Thanks to my parents and grandparents for their constant love and support. Last but not the least, for my loving and supportive husband, Dr. Bin Yang, his encouragement and support during the whole Ph.D. study are so appreciated.

Chapter 1

Introduction

Nowadays, demand for managing and querying interrelated but heterogeneous data sources is widespread. For example, each university employs an independent data source to record information of students, staff and courses. If a national education department conducts a survey of university employees on, say, salaries and working conditions, the department needs to query the data source of each university separately, which is inconvenient and time consuming. Such examples motivate the research conducted by the data integration community [HRO06] that users should be able to query different data sources via a unified interface rather than by visiting each of them separately.

Traditional data integration systems provide high-quality services but at a high cost [HRO06]. Before offering the integration services to users, precise mappings that describe the relationships of heterogeneous data sources are required in order to return accurate query results. However, the process of specifying mappings takes a great amount of manual effort [LN07, ATV08], thereby increasing the cost required to set up the data integration systems [Haa07] to quite a high level.

To reduce the high upfront cost of traditional data integration, *dataspaces* were proposed in 2005 [FHM05]. They aim at automatically setting up a data integration system, and meanwhile allow users to gradually improve the quality of the system. One element of the strategy to reduce the upfront cost is to automatically bootstrap the system, with the benefit of releasing experts from the tedious process of mapping specification. With the help of information that specifies the relationships between elements in different data sources, i.e., a degree of expressiveness equivalent to the schematic correspondences defined by Kim *et* *al.* [KS91], mappings can be automatically generated [MBPF09]. This constitutes a significant step towards automating the bootstrapping process of dataspaces [FHM05, HFM06]. The idea of automatically deriving schematic correspondences between data sources motivates the research described in this thesis.

The remainder of this chapter is organized as follows. In Section 1.1, we elaborate on the definition of schematic correspondences. We explain the process of traditional data integration in Section 1.2.1, along with the high-quality but also high-cost feature it exhibits. Next, in Section 1.2.2, we compare traditional data integration and dataspaces in order to motivate the need for automatic generation of views in dataspaces. We discuss the research background of inferring schematic correspondences in Section 1.3, and state the aim, objectives and contributions reported in the thesis in Section 1.4. Section 1.5 concludes by describing the structure of the remainder of this thesis.

1.1 Schematic Correspondences

Data sources usually refer to sources of digital data in the form of, e.g., databases, computer files or data streams. Apart from data, structured (and often semi-structured) data sources also contain schemas, or so-called metadata, which provide information about the data, such as their types, domain, time and date of creation, and authors.

Data sources that represent the same or similar real world information can exhibit great heterogeneity. For example, they may be represented using different database models, e.g., relational or XML models. It is also common that they are devised independently by different designers without coordination, whose views of the world differ even on the same objects. However, over time these data sources may come to be used together. Therefore, it is crucial to understand the heterogeneities among these data sources before we can really utilize and manage them together.

1.1.1 Definition

As mentioned above, a data source generally refers to a store of information rather than a specific source type. In this thesis, the research undertaken mainly addresses problems related with a particular type of data source, namely databases, and as such we will use the term *database* and the general name *data source* as synonyms.

A database consists of a collection of data records structured under some schema, where the underlying data captures the real world information in a special domain. Once a particular schema definition has been designed for a database, the associated data must comply with the schema. The schema specifies the structure of the data and the associated constraints using, for example, the relational data model, the object-relational data model, or the object-oriented data model [SKS02]. It also specifies the descriptive names, and the types of the underlying data. Unlike the structures of the data, which are constrained by the rules of the utilized data model, the names and types of the data could be fairly subjective to designers.

Heterogeneities or conflicts between databases mainly refer to different symbolic representations of data that represent the same real world information [KS91, KCGS93], mainly including two types:

- *data model heterogeneities*, which denote that data representing the same real world information are structured differently, for example, using different data models (e.g., relational and XML);
- *schematic heterogeneities*, which refer to the discrepancies between databases due to the use of different names or data types to describe the same real world information.

These two types of heterogeneities often appear simultaneously in real world applications, thus significantly increasing the complexity of reconciling them. For the purposes of this thesis, we chose as a representative schema definition, the relational schema, and address the problem of *schematic heterogeneities*. The problem of reconciling both *data model heterogeneities* and *schematic heterogeneities* is left for future work.

The term *schematic correspondence* refers to the linkage between the parts of schemas where a schematic heterogeneity exists, so we consider *correspondence* and *heterogeneity* as synonyms in this thesis.

1.1.2 Classification

Previous research yields several classifications of schematic correspondences between database schemas [Ler00, KS91, KCGS93]. In this thesis, we adopt the classification of schematic correspondences between relational schemas proposed by Kim *et al.* [KS91], which characterizes different symbolic representations of data that present the same real world information. In what follows, we have refined the characteristics of many-to-many entity type correspondences from [KS91] to distinguish horizontal and vertical partitioning. Before moving on to state the details, consider the schemas of two independently designed relational databases RDB1 and RDB2, where symbols * and $^+$ indicate primary key and foreign key attributes, respectively.

RDB1:
home_cust (id*, name, birth, a_id$^+$, p_city, p_area, p_local)
overseas_cust (id*, name, birth, a_id$^+$, p_city, p_area, p_local)
account (id*, name, balance, tax)

RDB2:
customer (key*, c_fname, c_lname, c_birth, account_key$^+$)
cust_phone (key*, city, area, local, extension)
cust_account (key*, account_name, account_balance)

Both RDB1 and RDB2 contain information about customers and their accounts, even though they represent the information differently. It can be identified that they represent broadly the same real world information, but that heterogeneities exist between them at both entity and attribute levels:
(i) *Entity-level correspondences* indicate the equivalence between two (sets of) entity types (i.e., tables), and can be decomposed into one-to-one and many-to-many entity type correspondences, where

- *One-to-one entity type correspondences* relate pairs of entity types that represent the same information. For example, account in RDB1 and cust_account in RDB2 can be considered to represent equivalent real-world notions but show the following heterogeneities:
    - *name conflict*, which indicates that the equivalent entity types have different names. In the following, this conflict is called Different Names for the Same Entity type (*DNSE*). When different entity types happen to have the same name, we call the conflict Same Name for Different Entity types (*SNDE*).

  - *missing attributes conflict*, which indicates attributes that are present in one entity type but not in the other (e.g., attribute tax in account is missing in cust_account).

- *many-to-many entity type correspondences* relate two sets of entity types that represent the same information. For example, in RDB1 home_cust and overseas_cust together describe the same information about customers as customer and cust_phone in RDB2. Note that these two sets of entity types do not have the same structure in RDB1 and RDB2, but the underlying information is similar. This difference in structure between each set of entity types results in distinct kinds of many-to-many conflict. Inheriting terms from distributed database systems [OV89], we define the structure within a set of entity types as:

  - *horizontal partitioning* (*HP*), where an original entity type is partitioned along its instances into new entity types. As such, all attributes of the original entity are present in each of new entity types (e.g., home_cust and overseas_cust in RBD1 are horizontal partitioning of customer information).
  - *vertical partitioning* (*VP*), where an original entity type is partitioned into new entity types whose attributes are subsets of the original entity. As such, some attributes are present in each of the new entity types, which are primary keys, whereas other attributes of the original entity types are present only once across all the new entity types (e.g., customer and cust_phone in RDB2 are vertical partitioning of customer information).

  Given the above partitioning information, we are then able to enumerate 4 types of many-to-many entity type correspondences: *HP* vs *HP*, *HP* vs *VP*, *VP* vs *HP* and *VP* vs *VP* correspondences. For example, the correspondence between entity type sets {home_cust, overseas_cust} and {customer, cust_phone} is a *HP* vs *VP* correspondence.

(ii) *Attribute-level correspondences* indicate the equivalence between two (sets of) attributes. For the remainder of the thesis, we assume that attributes associated by attribute-level correspondences belong to entity types that are participating in some entity-level correspondence. Similar to entity-level correspondences, the attribute-level correspondences can be decomposed into one-to-one and many-to-many correspondences as follows:

- *One-to-one attribute correspondences* relate pairs of attributes. Equivalent attributes may have different names, so such a conflict is called Different Names for the Same Attributes (*DNSA*) (e.g., attributes account.name in RDB1 and cust_account.account_name in RDB2). By contrast, attributes that are different but may have the same name give rise to Same Name for Different Attributes (*SNDA*) correspondences.

- *Many-to-many attribute correspondences* associate two sets of attributes that represent the same property of equivalent entity types. For example, both the single attribute home_cust.name in RDB1 and the set of attributes customer.c_fname and customer.c_lname in RDB2 represent names of customers.

Because of such schematic heterogeneities, it is hard for users to query different data sources using the same query expressions. This raises the need to offer users an integration system that reconciles heterogeneities between the data sources and provides a unified query interface, as presented in the next section.

1.2 Dataspaces: Next Generation of Data Integration

From a user's perspective, it is more convenient to issue queries against a single integrated schema than visiting a number of different data sources separately. However, as stated in Section 1.1, data sources may exhibit great diversity and heterogeneity, thus making it hard to use them together. In this section, we introduce two versions of a general solution for managing and querying heterogeneous data sources: traditional *data integration* in Section 1.2.1 and its evolution *dataspaces* in Section 1.2.2.

1.2.1 Traditional data integration

Data integration is a difficult challenge faced by applications that need to query across multiple autonomous and heterogeneous data sources. In a traditional data integration system, the data sources are usually referred to as *local sources* and the schemas are referred to as *local schemas*. The single interface through which users can access the data sources for queries is usually called a *mediated* (or *global*) *schema* [HRO06].

The initialization of a traditional data integration system generally requires the following two steps [Len02]: i) (semi-)automatically identifying matches that associate equivalent or similar parts between schemas; and ii) specifying views that explicitly describe the semantic mappings between elements of the mediated and the local schemas on the basis of the matches obtained in the previous step. Techniques for identifying matches and for specifying views are introduced in more detail in Chapter 2. When the user poses a query over the mediated schema, the query needs to be reformulated into a set of queries over the local schemas [HRO06] using the views. The query results are then evaluated, combined and returned to the user.

Several approaches have been proposed to define the views, including Local-as-View (LAV) [DGL00, FW97], Global-as-View (GAV) [GMPQ+97, ACPS96], and more recently Global-and-Local-as-View (GLAV) [FLM99, HIST03]. In contrast to GAV, where the mediated schema is presented as a view over the local schemas, LAV describes a local schema as a view over the mediated schema. Both methods have advantages and disadvantages, which we will discuss in Section 2.5, so GLAV was proposed in order to combine the benefits of GAV and LAV.

One of the bottlenecks in setting up a data integration system is the effort and knowledge required to specify views between the mediated schema and the local schemas, in order to return accurate query results. The knowledge required includes both an expertise in constructing the views (i.e., expressing them in a formal language) and domain knowledge about local sources (i.e., understanding the meaning of the schemas being mapped) [HRO06], and thus a human must be in the loop of specifying views [BM07].

1.2.2 Dataspaces

Dataspaces can be viewed as the next step in the evolution of traditional data integration [HFM06], as they share some characteristics with traditional data integration. Dataspaces, though being a new concept, have attracted significant attention [TIP10, MA10, DHY07, MCD+07, DS06, BDG+07, SDK+07, JFH08, SDH08, BEG+06, CKP08]. Most of these approaches have been classified in the survey [HBF+09], which also clarifies the representative key concepts in this area and characterizes the life cycle a dataspace management system may support.

Facing the same application scenarios as traditional data integration systems do, dataspace management systems are likewise required to offer query services over a large number of diverse and heterogeneous but interrelated data sources. Principally, a dataspace management system works in a "pay-as-you-go" fashion: "*in a sense, the dataspace approach postpones the labor-intensive aspects of data integration until they are absolutely needed*" [HFM06]. With this philosophy, the aim of a dataspace management system is to automatically offer integration services on demand at lower up-front cost with, as a consequence, potentially lower quality at the beginning. As "effort", e.g., in the form of feedback from users and developers, is incrementally put into improving the system, increased "quality" will become available to users, e.g., in terms of improvement to the accuracy of query results [BPE$^+$10].

A dataspace management system aims to release experts from the tedious process of mapping specification required to set up a traditional data integration system, and as such reduces the high upfront cost of traditional data integration. Thus, a dataspace management system can be defined as a data integration management system that offers two additional features: i) the system should be set up automatically; and ii) users are allowed to incrementally improve the quality of the system by annotating query results.

The research described in this thesis addresses the first feature of dataspaces, specifically, the feature of setting up a dataspace management system automatically. A step towards the automation of dataspace management systems is to automatically generate views, as proposed by Mao *et al.* [MBPF09]. The authors argue that "*the classical approach of correspondence identification followed by (manual) mapping generation can be simplified through the removal of the second step by judicious refinement of the correspondences captured*". This judicious refinement of the correspondences defined in the paper refers to the schematic correspondences we have introduced in Section 1.1.2. Thus, since the kind of expressive information captured by schematic correspondences between two schemas is so important for automatic view generation, which in turn is crucial for bootstrapping dataspaces, is it possible to automatically infer such schematic correspondences between schemas? This thesis contributes one approach to solving this problem.

Before closing this section, we also need to mention that automatic view generation is not only applicable to bootstrapping dataspaces, but is also suitable in some other applications, such as data exchange. *Data exchange* declaratively specifies mappings that transfer and restructure data under one schema (called a source schema) into instances of a different schema (a target schema) [FHH$^+$09, FKMP03]. It is used in tasks where data needs to be transferred between independently devised data sources. Despite this being a different use from views in data integration that reformulate queries, the mappings in data exchange are still a type of view. As such, the process of automatic view generation can also be used in data exchange to reduce the amount of manual effort involved. Hence, to generalize the problem of view generation, we do not particularly emphasize the role of views in data integration that exist between the mediated schema and several local schemas, but consider them prevalent between any two schemas.

1.3 Research Context

Various kinds of associations that describe the semantic relationships between two schemas can be specified, such as matches identified by schema matching approaches, views specified manually or inferred semi-automatically, or schematic correspondences we refer to in this thesis. In this section, we introduce different levels of semantic relationships between two schemas that have been proposed by others, with the aim of demonstrating the novelty of the research conducted in this thesis.

Bernstein *et al.* [BM07] summarized three types of semantic relationships between two schemas where information carried is successively refined from matches to views and to transformations.

- Matches, the lowest level of semantic relationships, are defined as pairs of elements between two schemas that are believed to be related in some unspecified way. Examples of matches are presented in Figure 1.1 using the relational schemas RDB1 and RDB2 introduced earlier in Section 1.1.2. In Figure 1.1, each line (i.e., a match) relates a pair of elements (e.g., tables and attributes) between the two schemas that show certain similarity, e.g., in their names, data types, instances or structures, and is labeled with a score to indicate the degree of similarity, usually in the range between [0, 1]. A higher similarity score indicates that the related elements are more

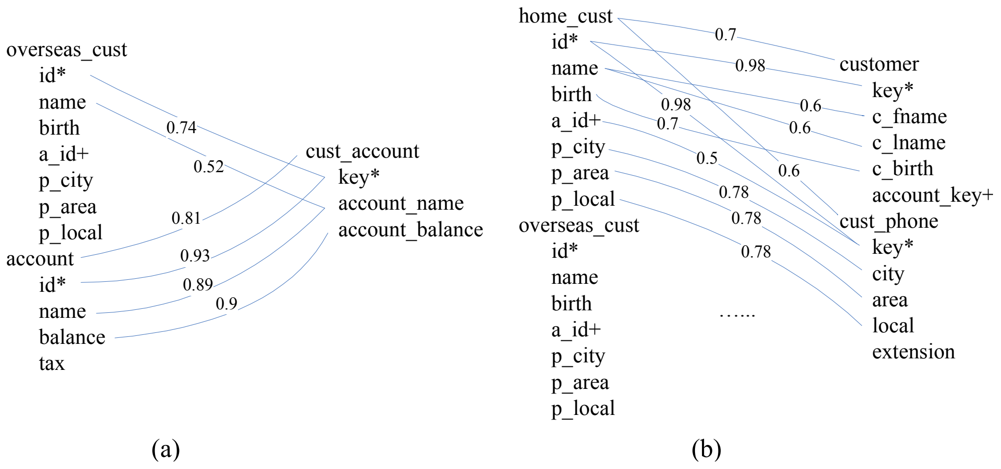


Figure 1.1: Examples for schema matching results.

similar. Figure 1.1 only shows a subset of matches between the two schemas, as we omit some others for simplicity. For example, in Figure 1.1(b), the matches between overseas_cust and customer and between overseas_cust and cust_phone are similar to the matches between home_customer and customer and between home_cust and cust_phone, and thus are omitted.

These matches only indicate that elements are associated between the two schemas and could be used in the process of view generation. They do not specify a view. Popular schema matching techniques for identifying matches between elements include comparison of, e.g., names, data values or data types [DR07, MGMR02, DDH01, DLD$^+$04, XE06], as will be discussed in detail in Chapter 2. Some benchmarks for evaluating schema matching systems [DBH07, EFH$^+$09] have been developed to assess the accuracy of their identified matches based, e.g., on precision, recall and F-measure. However, few researchers have ever asked: what is the meaning of a match? How much useful information is carried by a match that can be employed to generate views? How can a benchmark be devised to evaluate matches in terms of the way they associate elements rather than their accuracy?

- Views, an intermediate level of semantic relationships, have been mentioned earlier in Section 1.2. They are defined as a declarative specification of relationships between instances of two schemas. Various approaches (e.g., Clio [FHH$^+$09]) have been proposed for (semi-)automatically generating views with the help of matches, a task known as schema mapping. In

Chapter 2, we define three kinds of views, namely Local-as-View (LAV), Global-as-View (GAV), and Global-and-Local-as-View (GLAV), and discuss schema mapping techniques in more detail.

In general, views are generated by interpreting the matches. For example, given the matches shown in Figure 1.1(a), the interpretations that account and cust_account are equivalent and that overseas_cust and cust_account are equivalent would give rise to Views *1* and *2* below. However, not every generated view captures the correct meaning, e.g., View *2*, and as such needs to be removed manually.

View 1:
SELECT id, name, balance FROM account =
SELECT key, account_name, account_balance FROM cust_account

View 2:
SELECT id, name FROM overseas_cust =
SELECT key, account_name FROM cust_account

Based on the matches presented in Figure 1.1(b), the interpretations, such as i) home_cust and customer are equivalent, ii) overseas_cust and customer are equivalent, or iii) {home_cust, overseas_cust} is equivalent to {customer, cust_phone}, would give rise to Views *3* to *5*. Among them, Views *3* and *4* only partially reflect the real world information. View *5* precisely specifies the relationship between the elements, but has to be generated using additional knowledge (e.g., integrity constraints), in order to determine that home_cust and overseas_cust should be unioned, and that customer should be joined with cust_phone on their primary keys key [FHH+09].

View 3:
SELECT id, name, birth, a_id FROM home_cust =
SELECT key, concat(c_fname, c_lname), c_birth, account_key FROM customer

View 4:
SELECT id, name, birth, a_id FROM overseas_cust =
SELECT key, concat(c_fname, c_lname), c_birth, account_key FROM customer

View 5:
SELECT id, name, birth, a_id, p_city, p_area, p_local
FROM home_cust
UNION

```
SELECT id, name, birth, a_id, p_city, p_area, p_local
FROM overseas_cust
=
SELECT A.key, concat(A.c_fname, A.c_lname), A.c_birth, A.account_key, B.city,
B.area, B.local,
FROM customer as A, cust_phone as B
WHERE customer.key = cust_phone.key
```

As can be observed, human has to be involved in the loop of specifying views, because there is a greater likelihood that incorrect or incomplete interpretations of matches give rise to incorrect or partial views, thereby implying the need for manual effort to remove them, e.g., Views *2*, *3* and *4*. Furthermore, additional knowledge is required to generate accurate views, e.g., View *5*, which may not always be available. These are due to the fact that matches only associate elements by assigning them a similarity score based on names and data values, but cannot provide sufficient information required by schema mapping approaches.

- Transformations, or so-called functional views, are queries that can be utilized by certain runtime environment that specify the highest-level semantic relationships between the two schemas. This process takes as input views and produces queries in the form of, e.g., Xquery or XSLT. Representative examples for generating transformations are found in the publications of Clio project [FHH$^+$09]. The process of developing transformations is beyond the scope of this thesis. Therefore, we will not elaborate on it further.

The principle contribution of this thesis, i.e., inferring schematic correspondences between two schemas, bridges the gap between identifying matches and specifying views. Schematic correspondences differ from views in that they do not aim to declaratively specify a relationship expression. On the other hand, they tend to offer more information than the matches identified by the existing schema matching approaches, as required by automatic view generation. For example, a schematic correspondence explicitly indicates that its associated elements represent the same real world information, and thus (in our example above) overseas_cust and cust_account will not be related. Furthermore, a schematic correspondence also specifies the equivalence between two groups of elements, e.g., {home_cust, overseas_cust} and {customer, cust_phone}, with partitioning information, e.g., horizontal and vertical partitioning, respectively. Consequently, a horizontal partitioning indicates that the elements in a group should be unioned, and a vertical partitioning indicates that they should be joined. Thus, if such information is available, View *5* (in our example) should be generated automatically [MBPF09].

1.4 Thesis Aims, Objectives and Contributions

As described in Section 1.2.2, one of the distinguishable features of dataspaces is to automatically set up a dataspace system. This feature motivates the research presented in this thesis. However, as manual effort has to be required to specify views between data sources using the state-of-the-art techniques (see Section 1.3), a novel method for inferring a different type of relationships between data sources to bootstrap the setup process of a dataspace system shows great influence to dataspaces. Thus, this thesis aims to investigate the identification of schematic correspondences between interrelated but heterogeneous data sources, which are expressive enough to underpin algorithms for automatically generating views with a view to reducing the manual effort involved in setting up a dataspace system.

We follow a two-step research programme to achieve the overall aim of the thesis. In the first step, we develop a benchmark to investigate whether or not the existing schema matching systems are effective in inferring schematic correspondences, and thus we can diagnose whether or not the systems can be used by the method for automatically specifying views [MBPF09]. In the second step, we devise a novel method for inferring schematic correspondences based on the lessons learnt from the first step. In principle, we address the following objectives:

- To develop a benchmark that identifies, describes and empirically evaluates the effectiveness of existing schema matching approaches for inferring schematic correspondences between data sources. Diagnosing the problems exhibited in current techniques allows us to understand their shortcomings fundamentally, and, thus, facilitates devising a method for inferring schematic correspondences.

- To devise an inference technique that enables the characterization of correspondences at a level suitable for automatic view generation with quality measures, and to evaluate the resulting proposal.

Given the aim and the objectives, the main contributions of this thesis are the empirical evaluation of the existing schema matching approaches and the technique for automatically inferring schematic correspondences. In terms of the empirical evaluation, we have developed a benchmark, called MatchBench, thereby making the following contributions:

- A collection of synthetic scenarios that manifest various types of schematic heterogeneities of Kim *et al.* [KS91]. MatchBench offers four scenario spaces, where the amount of heterogeneity between pairwise schemas in each scenario systematically varies: a collection of scenarios with one-to-one equivalent but heterogeneous entity types, a collection of scenarios with different entity types having some similarities between them, a collection of scenarios with many-to-one attribute correspondences, and a collection of scenarios with many-to-many equivalent but heterogeneous entity types.

- An experimental design over the above scenarios that investigates the effectiveness of existing schema matching approaches in diagnosing schematic heterogeneities. We have designed experiments that can be categorized into positive and negative test cases. Given a particular type of schematic correspondence, the corresponding positive experiment selects scenarios where the current heterogeneity is present and evaluates the performance of matchers in diagnosing it. In contrast, the corresponding negative experiment runs the matchers on scenarios where such a heterogeneity is absent and evaluates whether the matchers are able to not report the heterogeneity where they should not do so. In total, we cover *DNSE* (Different Names for the Same Entity types), *DNSA* (Different Names for the Same Attributes), *missing attributes*, *many-to-one attributes*, and *many-to-many entity types* conflicts.

- An empirical study of three well-known schema matching platforms, namely COMA++ [DR07], Rondo [MRB03] and OpenII [SMH$^+$10] using MatchBench. These platforms have been used in the evaluation because they are among the best known schema matching platforms, and are publicly available. We investigate the advantages and disadvantages exhibited in these platforms and summarize the lessons learnt from the empirical study with a view to developing the method for inferring schematic correspondences.

In terms of the technique for inferring schematic correspondences, we have conducted a two-step approach that firstly infers the schematic correspondences at the entity-level and secondly identifies attribute-level schematic correspondences for each resulting entity-level schematic correspondence. In particular, the following contributions have been made:

- An evolutionary search method, specifically a genetic algorithm, that infers entity-level schematic correspondences between source and target schemas using matches provided by existing schema matching approaches. We define a solution as a set of entity-level relationships (*ELRs*), which associate pairwise entities/entity sets between the source and target schemas. We apply the genetic algorithm to search for a particular solution that satisfies the requirements of entity-level schematic correspondences. Applying the search method allows different solutions to compete with each other, and thus does not require heuristic rules, e.g., thresholds, to select the final result. We have designed *phenotype* and *genotype* representations of a set of *ELRs*, and have implemented the various operators required by the genetic algorithm, including an objective function used for evaluating solutions and for guiding the search process. A set of *ELRs* that is assigned the highest fitness value by the objective function is considered as the entity-level schematic correspondences, based on which we further identify a set of attribute-level relationships (*ALRs*) as the schematic correspondences at the attribute-level.

- An objective function that models the requirements for identifying entity-level schematic correspondences and calculates the relative fitness value of a solution within the search space. The requirements include that: i) each entity-level schematic correspondence represents the equivalent relationship between two (sets of) entities, implying that two different (sets of) entities that coincidentally have overlapping information should not be associated; ii) equivalent n-to-m entities should be associated by an entity-level schematic correspondence rather than its subsets of entities, e.g., (n-1)-to-m entities; iii) each entity-level schematic correspondence that associates n-to-m entities is able to establish their specific partitioning types (i.e., horizontal and vertical partitioning); and iv) the more entity-level schematic correspondences that satisfy requirements i) - iii) that are identified the better. Given a solution composed of a set of *ELRs*, the objective function calculates a similarity score for each *ELR* using the vector space model [SWY75], and aggregates these similarity scores as the fitness value of the solution.

- An empirical evaluation that assesses the effectiveness of our approach for inferring schematic correspondences. In particular, we compare our method to COMA++ [DR07] using experiments provided by MatchBench and relational databases representing various schematic heterogeneities offered by the Amalgam benchmark [MFH^{+}01]. We decided to make a comparison with COMA++ only because it has already been shown to perform better than the other two platforms, i.e., Rondo [MRB03] and OpenII [SMH^{+}10], in an application of the MatchBench benchmark. The experimental results show that our method is more effective (i.e., higher Precision, Recall and F-measure) in inferring schematic correspondences than COMA++.

1.5 Thesis Structure

The remainder of this thesis is structured as follows. We discuss the technical details of schema matching and view generation operators in Chapter 2 with the aim of providing a general background for the research described in this thesis. In Chapter 3, we introduce the synthetic scenarios offered in MatchBench and the experiments designed for evaluating existing schema matching approaches for diagnosing schematic correspondences. We discuss the application of MatchBench to three well-known matching platforms, namely COMA++ [DR07], Rondo [MRB03], and OpenII [SMH^{+}10] in Chapter 4. We present the method for inferring schematic correspondences, including the evolutionary search framework for searching entity-level relationships (*ELRs*), representations of the *ELRs*, the objective function and the method for inferring attribute-level schematic correspondences, in Chapter 5. We report on an experimental evaluations of our approach for inferring correspondences in Chapter 6. We review the most significant contributions reported in this thesis in Chapter 7, and we also discuss some of the remaining open research issues.

Chapter 2

Schema Matching and View Generation

Schematic correspondences specify relationships between elements in two schemas. They provide richer semantic information than matches, and aim to support the bootstrapping of dataspaces by enabling the process of automatic view generation, as introduced in Chapter 1. In this chapter, we present a background research on inference of schematic correspondences, including a detailed description and classification of schema matching approaches that identify matches, and an introduction to techniques for view generation. We mainly focus on presenting schema matching approaches in this chapter, because their techniques form the foundation for inferring schematic correspondences, and their results are used as the input for the inference process. We also present issues in view generation to illustrate the difficulty to directly generate views from matches, thus highlighting the importance of being able to infer schematic correspondences that can then be used for the automatic generation of views.

The remainder of this chapter is organized as follows. In Section 2.1, we define the schema matching operator along with its input and output, and present a general classification of matching techniques. Following one of classification categories, we present schema-level and instance-level matching techniques in Sections 2.2 and 2.3, respectively. In Section 2.4, we summarize some of the state-of-the-art schema matching approaches. We introduce schema mapping techniques in Section 2.5 and conclude the chapter in Section 2.6.

2.1 Schema Matching Definition and Classification

Although the outcome of schema matching approaches has given several names, e.g., matches [DR07, MGMR02, DLD$^+$04], mappings [RB01, HRO06, DDH01] and value correspondences [MHH00], they mainly represent the same type of semantic relationship between two schemas, i.e., the matches introduced in Section 1.3. In this section, we elaborate on the definition of the schema matching operator, its inputs and outputs, and provide a technical classification.

2.1.1 Definition

The schema matching operator takes as input two data sources, usually referred to as the *source schema* and the *target schema* [RB01, SE05]. Each schema needs to conform to a data model, e.g., the relational model, XML model or Ontology [DR07, TC07, GSY04, CFM06]. We define results of a schema matching operator as a collection of 4-tuple matches $\langle E_1, E_2, R, S \rangle$, each of which indicates that elements (e.g., tables or attributes in the relational model) of the source and target schemas are associated in a particular manner, where

- E_1 is a set of elements from the source schema;
- E_2 is a set of elements from the target schema;
- R is an expression that specifies the relationship between the associated elements E_1 and E_2, and may be equivalence ($\equiv$) or more general ($\sqsupseteq$), etc.;
- S is a score that indicates the similarity degree between E_1 and E_2 the matching operator provides given the relationship R identified.

We define the schema matching operator as a function that takes as input two schemas and control parameters, and produces a collection of 4-tuple matches between them. For example, given the relational databases RDB1 and RDB2 presented in Section 1.1.2, the schema matching operator produces a collection of matches graphically described in Figure 1.1 in Section 1.3. The association between account.id and cust_account.id in Figure 1.1(a) can be defined as a 4-tuple match $\langle$account.id, cust_account.id, $\equiv$, 0.81$\rangle$.

2.1.2 Technical classification

Schema matching techniques can be classified along three criteria: input, output and technical parameters, as shown in Table 2.1.

Input	
Schema type	XML
	Relational
	Ontology
Number of Schemas	1:1
	1:n
Schema Scale	Large
	Medium
	Small
Internal Representation	Direct labelled graph
	Hierarchy tree
	Conceptual model

Output	
Type of result	Equivalence
	Semantics
Type of score	Similarity coefficients
	Probabilities

Technical Parameters	
Granularity	Element-level
	Structure-level
Matching level	Schema-level
	Instance-level
Reuse	Reuse
	Non-Reuse
Base	Lexical
	Constraint
Combination	Hybrid
	Composite
Cardinality	1:1
	1:n
	n:1
	n:m
Auxiliary information	Dictionaries
	Thesauri

Table 2.1: A Classification of Schema Matching Techniques.

Input describes information related to the source and target schemas that is required by a given schema matching approach, and can be categorized as follows:

- *Schema type* mainly specifies the schema definition of data sources that a given matching approach is able to support. For example, some approaches, e.g., [BN05, DLD$^+$04, DR07, DDH01, KN03, KN08, BEFF06, WT06, DKS$^+$08], identify matches between relational schemas; some methods, e.g., [DR07, MBR01, TC07, ASS09, GYS07, TC07], match XML schemas; and some others, e.g., [DMDH02, TLL$^+$06, PDYP05, CFM06, ZLL$^+$09, HQC08, TLL$^+$06], align elements between ontologies. Given different types of schemas, the basic techniques for matching schemas, such as comparing their element names, may be similar, but specific techniques are usually required during the match process to handle distinct schema structures.

- *Number of schemas* refers to the number of source and target schemas that a given matching approach can match. Usually, matches are between one source and one target schema [DR07, BEFF06, HQC08]. Some approaches, e.g., LSD [DDH01], match elements between a mediated schema and several local schemas in traditional data integration scenarios.

- *Schema scale* describes the size of the input schemas, as special techniques may be required to match large scale schemas when considering the efficiency of a given matching approach, as addressed by COMA++ [DR07] and Falcon [HQC08].

- *Internal representation* denotes the data structure into which the schemas are internalized, and over which the match operation is performed. Some methods, e.g., [DR07, MBR01], translate the source and target schemas, which may be defined differently in terms of the *schema type*, into an internal representation before matching them. Some approaches, e.g., [MGMR02, XE06], do not explicitly state the *schema types* they support, and take as input data sources structured under the internal representation. Others, e.g., [BN05, DLD$^+$04], directly match schemas without using any internal representation. Typical internal representations include the directed graph models supported by, e.g., Similarity Flooding [MGMR02] and COMA++ [DR07], the hierarchical trees used by, e.g., Cupid [MBR01], and conceptual-model graphs used by, e.g., Xu *et al.* [XE06].

Output describes information returned as result of the matching process, giving rise to the following classification:

- *Type of result* describes the semantic relationship postulated by a match. Most schema matching approaches, e.g., [DLD$^+$04, KN08, TLL$^+$06, DR07], postulate an equivalence relationship ($\equiv$). A few others are able to provide more expressive results, such as more general ($\sqsupseteq$), less general ($\sqsubseteq$) and incompatible ($\perp$) [GYS07], or *Has-a* and *Is-a* [WP08].

- *Type of score* expresses the nature of the score that represents the strength of the matches and usually lie in the [0, 1] interval.

Technical Parameters classify the various properties exploited by the state-of-the-art schema matching approaches as follows.

- *Schema or instance*: some methods only consider schema-level information during matching (e.g., element names and data types); others consider instance-level information (i.e., data content).

- *Element or structure*: matching methods may identify matches between individual schema elements (e.g., attributes) or between combinations of elements that appear as complex structures (e.g., the path from the root element to the matched element).

- *Reuse or non-reuse*: matching approaches may exploit and reuse information from previous experience to improve the output, e.g., previous matching results and corresponding user feedback.

- *Lexical or constraint*: a matcher may use a lexical approach, such as comparing names and textual descriptions of schema elements, or a constraint-based approach that utilizes constraints of two schemas, such as comparing primary key constraints or unique constraints.

- *Composite or hybrid*: a matcher may combine distinct match functions (such as comparing element names, data types or instances) differently. A composite matcher combines results (i.e., matches) of other matchers, each of which implements a specific function. A hybrid matcher implements various functions inside and returns matches that are comparable to results of a composite matcher.

- *Matching cardinality*: matching approaches may relate different numbers (i.e., one or more) of elements between source and target schemas. Thus, we can enumerate four cardinalities of associated elements: 1:1, n:1, 1:n, n:m.

- *Auxiliary information*: some matching methods make use of auxiliary information, such as dictionaries and thesauri (e.g., WordNet [Mil95]).

A more comprehensive description will be presented in Sections 2.2 and 2.3.

2.2 Schema-Level Matching

Schema-level matching only makes use of schema information to identify matches between source and target schemas [DR07, MBR01, MGMR02, TC07]. Depending on the expressiveness of schema languages and the chosen internal representation, the available information usually includes properties of schema elements, such as names, descriptions, data types and constraints (e.g., integrity and referential constraints); relationships between schema elements, such as *part-of* and *is-a*; and schema structures, such as relational and XML models. Table 2.2 provides a classification of schema-level matching techniques. In particular, the techniques are categorized into element-level and structure-level matching.

Category			Method Description
Element-level	String-based		Affix, Edit distance, N-gram, etc.
	Lexical		Tokens, Stemming, Elimination, etc.
	Semantics-based		Semantic relations (e.g., Synonyms)
	Constraint-based		Data types, keys constraint, etc.
	Domain-based		Domain thesauri
Structure-level	Scope	Global	Considering the complete schema
		Local	Considering parts of the schema
	Neighborhood	Distinctive	Serving different roles
		Ambiguous	Serving the same roles
	Traversal Strategy	Top-down	Considering descendants when matching
		Bottom-up	Considering ascendants when matching

Table 2.2: A Classification of Schema-Level Matching Techniques.

2.2.1 Element-level matching techniques

Element-level matching techniques compare properties of single elements in isolation, regardless of their relationships with other elements. These techniques are usually used before the structure-level techniques, and as such serve as the foundation for matching element structures. The major element-level techniques are classified as follows:

String-based techniques compare name and description strings of elements by considering their lexical structures. Such comparisons usually express the distance between the two strings through a similarity coefficient, where a greater value indicates more similar strings. In particular, the techniques exploited by the schema matching approaches [DR07, NM01, GYS07, MGMR02, MBDH05] include:

- *Affix* comprises prefix and suffix identifications. The prefix comparison checks two strings from their beginnings; suffix, on the other hand, compares their endings. For example, the prefix comparison will assign a greater value to *int* and *integer* than to *phone* and *telephone* because the former pair has the same beginning, while the suffix technique will do the opposite.

- *Edit distance* computes the similarity between two strings by counting the number of edit operations (i.e., insertion, deletion and substitution) required to transform one into the other, and is usually normalized by the length of the longer string. For example, the edit distance between strings *empl* and *employee* is 0.5.
- *N-gram* calculates the number of shared *n-grams* (i.e., sequences of n characters), and is normalized by the cardinality of the longer *n-gram* set. For example, *addr* has two 3-gram {add, ddr} and *address* has five 3-gram {add, ddr, dre, res, ess}, and both share the 3-gram set {add, ddr}. Normalized by the cardinality of the largest 3-gram set, their 3-gram similarity score is 0.4.

Lexical techniques exploit natural language processing methods to parse the name and description strings of elements [GYS07, MBR01, TC07, TLL$^+$06, ASS09, IIK08], with the purpose of preprocessing these strings before applying the string-based techniques, and are classified as follows.

- *Tokens* are parsed from a string by a tokenizer that cuts the original string at punctuation, upper cases, special symbols, digits, etc.. For example, *author_names* can be parsed into the tokens *author* and *names*.
- *Stemming* algorithms identify the basic form of a word by removing its variations (e.g., plural, past tense). For example, *names* is stemmed into *name*.
- *Elimination* discards tokens, e.g., articles, prepositions, conjunctions, in order to avoid comparing strings that are known not to denote real-world concepts.

Semantic-based techniques estimate similarity of two elements from the meaning denoted by their names [PS11, GYS07, XE06, IIK08, DMDH02, CFM06]. In contrast to string-based and language-based techniques that only rely on the character sequence of strings, the semantic-based techniques consider the semantic relationships of two strings (e.g., S_1 and S_2), such as synonymy (i.e., S_1 and S_2

describe the same object), hyponymy (i.e., S_1 is a kind of S_2) or hypernymy (i.e., S_2 is a kind of S_1). For example, strings 'postcode' and 'zipcode' are synonyms according to WordNet [Mil95], and as such may be deemed equivalent. This approach usually requires auxiliary sources, such as dictionaries and thesauri (e.g., WordNet [Mil95]), in order to identify such semantic relationships.

Domain-based techniques apply thesauri that store specific domain knowledge to improve the accuracy of string comparisons [DR07, MBR01, GYS07]. For example, they could help to recognize multi-word strings (e.g., firstname = fname), acronyms (e.g., PO = PurchaseOrder) and abbreviations (e.g., addr = address).

Constraint-based techniques make use of data modeling constraints to match individual elements that declare, for example, data types, value ranges, uniqueness, characteristics of key attributes (e.g., primary and foreign) and cardinalities [TLL$^+$06, DR07, CFM06, ASS09].

2.2.2 Structure-level matching techniques

Structure-level matching techniques identify matches between two schemas by comparing combinations of elements that appear together in a structure. The structural similarity of two elements is derived from their element-level similarity, complemented with an analysis of their positions in the schemas and the combined similarities of their neighbour elements. We refer to such a process as *similarity propagation*, during which similarities from neighbour in elements are propagated to the matched elements, using various strategies, as described in the following.

Scope represents the range of schema elements over which the structure-level techniques operate, because it is not always necessary to include the whole schemas during matching. A *global* strategy always includes the complete set of schema elements in the structural matching [MGMR02, TC07, DR07, TLL$^+$06, MBR01]. In contrast, a *local* strategy only considers partial schema elements, and is found helpful when dealing with large scale schemas [HQC08, XE06, DR07], as it usually decomposes the whole schemas into reasonably small segments and then matches them against each other.

Neighbourhood refers to the nearby elements whose similarity may influence the matched elements. Sometimes, nearby elements in a special position (e.g., parents, children and siblings), called the *distinctive* neighbours, have a unique influence on the matched elements. The similarities of the *distinctive* neighbours will be combined in different ways to the matched elements [DR07, TC07, MBR01, ASS09]. Other approaches, e.g., Similarity Flooding [MGMR02], combine similarities of neighbour elements in any position (e.g., parents, children and siblings), called *ambiguous* neighbours, in the same way to the matched elements.

Traversal strategy concerns the order in which similarities between elements are explored. For example, a *top-down* method explores similarities from root elements to leaf elements and a *bottom-up* method aggregates similarities of children to a higher-level similarity. Usually, structure-level techniques consider similarity propagation in both directions. The top-down method combines similarity of two elements down to their children when they are considered similar enough, and the children's similarities are in turn propagated to their own children recusively. It usually involves a pruning process, which no longer passes the similarity down if two elements are considered different. The bottom-up method works in the opposite direction but is more expensive than the top-down method as it works without pruning. On the other hand, it can also achieve more precise results [MBR01] by gradually combining the lowest-level similarities into the higher-level similarities, as the lower-level similarities that might be pruned in the top-down method are used by the bottom-up method.

The advantage of schema-level techniques lies in their use of direct and simple properties of schema information. However, although different structure-level techniques are applied, most schema matching approaches only identify simple one-to-one matches between individual elements [DR07, MBR01] rather than associating two sets of elements, and thus are not particularly helpful to discover more complex many-to-many schematic correspondences.

2.3 Instance-Level Matching

Instance-level techniques exploit the underlying data content of source and target schemas to perform the matching tasks. Sometimes, these techniques are designed and implemented with the purpose of complementing the schema-level techniques, thereby helping to improve the overall matching accuracy (i.e., precision and recall) [EM07]. In special scenarios where the schema information exists but cannot be used by the schema-level techniques (e.g., elements have opaque names), only instance information can be utilized during matching [KN03, KN08]. Some approaches [DLD$^+$04, WT06, DKS$^+$08, XE06] identify complex attribute matches, e.g., concat(first-name, last-name) = name, by investigating the linkage between attribute instances. Others [XE06, TLL$^+$06] try to reconcile the data conflicts (e.g., 06/2010 and June 2010) in order to match equivalent attributes.

The way that instances are utilized during the matching may also be different. Most approaches, e.g., [DMDH02, DDH01, EM07, DLD$^+$04, WT06, DKS$^+$08, XE06, TLL$^+$06, BN05], match elements by directly comparing their instances. Some earlier works, e.g., [DMDH02, DDH01], apply machine learning techniques to train matchers using a subset of instances before matching elements. Other methods, e.g., [KN03, KN08], match schema graphs that model the interdependencies among elements, which are constructed by analyzing the relationships of element instances.

Most instance-level approaches identify matches between the lowest-level elements that are directly associated with data content (e.g., attributes) rather than between higher-level elements (e.g., tables in relational model), because the *similarity propagation* techniques presented in Section 2.2.2 can be applied to infer the higher-level similarities from lowest-level ones. Similar to schema-level techniques, instance-level approaches identify one-to-one matches [DMDH02, DDH01, BN05, KN03, KN08, EM07, XE06] and many-to-many matches [DLD$^+$04, WT06, DKS$^+$08, XE06, TLL$^+$06, XE06], and can be classified along the following dimensions:

- *Pattern-based* matching requires formulae before the matching process can be carried out. The formulae are used to transform instances between two sets of attributes. Usually, several different types of formulas are provided, each of which serves a special matching purpose, such as *concat*(*att-i*, *att-j*) = *att* (e.g., concat(first-name, last-name) = name) and *att-i*/*att-j*/*att-k* = *att* (e.g., day/month/year = date). Any two sets of attributes from the source and target schemas whose instances satisfy a particular formula can be identified as being equivalent. The matching of attributes is then considered to be a process of searching and validating attributes that satisfy the formulae. In some instance-level approaches, formulae are manually specified with the help of domain knowledge [XE06, TLL+06, DLD+04, WT06, DKS+08]; others derive the formulae by training with given data [DMDH02, DDH01].

- *String-based* matching compares values of attributes, but does not differentiate their data types (e.g., numeric, char), taking each value as simply a sequence of characters. String-based techniques presented in Section 2.2.1 are, therefore, also suitable for matching attribute instances. In general, there are two methods to compare instance values, namely *vertical* and *horizontal* comparisons. The *vertical* instance comparison technique derives the instance similarity of two attributes by comparing their data values [EM07], whereas the *horizontal* instance comparison technique is only applied to compare attribute instances between relational databases [BN05], which first computes similarities of tuples between two tables and then derives the similarity of two attributes from the similarities of tuples.

2.4 State-of-the-Art Schema Matching Systems

In this section, we compare the characteristics of six state-of-the-art schema matching systems (see Table 2.3), in terms of the schema matching techniques described earlier in this chapter. Note that average F-measure at the bottom of Table 2.3 directly comes from the original papers, and therefore, the test cases used by these systems are different. We also describe their matching procedures by discussing how such techniques are utilized. It is noticeable that some approaches tend to focus only on either schema-level matching (e.g., Similarity Flooding [MGMR02]) or instance-level matching (e.g., iMAP [DLD+04], Dumas [BN05] and LSD [DDH01]), while others (e.g., COMA++ [DR02, DR07, EM07] and Xu *et al.* [XE06]) try to use both kinds of information, if available. Specifically,

- COMA++ [DR07] is a schema matching platform that provides a combination of schema-level and instance-level matching supported by a library of matchers. It extends COMA [DR02] to include more schema-level matchers and matching strategies, and complements it with instance-level matching [EM07]. COMA++ identifies 1-to-1 matches between elements of two

Matchers		COMA++ [DR02] [DR07] [EM07]	SF [MGMR02]	iMAP [DLD+04]	Dumas [BN05]	LSD [DDH01]	Xu *et al.* [XE06]
Schema types		Relational, XML, Ontology	Relational, XML, ...	Relational	Relational	XML	—
Number of schemas		1:1	1:1	1:1	1:1	1:n	1:1
Schema scale		Small, Medium, Large	Small	Small	Small	Small	Small
Internal Representation		Directed graph	Directed labelled graph (OIM) [BBC+99]	—	—	—	Conceptual model (OSM-L) [Emb97] [LEW00]
Schema-level	Element-level	String, Semantic, Constraint, Domain	String	—	—	—	Semantic -based
	Structure-level	Global & local, Distinctive, Top-down & Bottom-up	Global, Ambiguous,	—	—	—	Global & local, Distinctive, Top-down
Instance-level	Patter-based	—	—	Manual formula	—	Training formula	Manual formula
	String-based	Vertical	—	—	Horizontal	—	Vertical
Reuse		Previous results	—	—	—	—	—
Combination		Composite	Hybrid	Composite	Hybrid	Composite	Hybrid
Cardinality		1:1 n:1 n:m	1:1	1:1, n:1, n:m	1:1	1:1	1:1, n:1, n:m
Auxiliary information		Dictionaries of synonyms & abbreviations	—	Domain constraints	—	Domain constraints	Domain Knowledge, WordNet [Mil95]
Accuracy (Average F-measure)		75%	55%	65%	73%	78%	96%

Table 2.3: Characteristics of the State-of-the-Art Schema Matching Systems.

schemas. Before carrying out the matching procedure, it constructs a directed graph as the internal representation for each schema, in order to support matching between different schema types (e.g., relational, XML schemas, or ontologies).

As a matcher library, COMA++ provides schema-level element matchers, e.g., name and data type matchers, schema-level structural matchers, e.g., parents (top-down propagation) and leaves (bottom-up propagation) matchers, and instance-level matchers (vertical comparison). In addition to the element-level matchers, COMA++ also supports the lookup of synonyms and abbreviations. COMA++ offers choices for combining results of different matchers (composite combination), including *average* (with equal weight to results of different matchers), and *max/min* (always returning matches with the maximum/minimum similarity scores).

On top of the matchers, matching strategies are available, including *All-Context*, which matches paths from the root to nodes in hierarchical data sets; *NoContext*, which only considers single nodes during matching; and *FilteredContext*, which seeks to match paths of nodes only when the nodes are identified as being similar. COMA++ claims to be able to match large scale schemas efficiently. This is achieved by first segmenting the large schemas into element fragments and then matching fragments only if their root elements are identified as being similar. COMA++ also supports the refining of previous match results into new results, a process called reuse, where the n-to-1 (1-to-n) and n-to-m matches are produced by simply merging the 1-to-1 matches together that are associated with the same elements. Note that these are different from the n-to-m schematic correspondences we have discussed in Section 1.1, where the n (m) elements that appear together indicate an internal relationship (e.g., the n elements are horizontally partitioned).

- Similarity Flooding (SF) [MGMR02] is a schema-level matching method, which applies a different structure-level matching technique from COMA++. It identifies 1-to-1 matches between elements of two schemas, each of which is imported into an internal graph model supported by Microsoft Open Information Model (OIM) specification [BBC+99], for the benefit of matching different types of schemas. Similarity Flooding applies a hybrid combination technique: initial matches are produced at the element-level, and these matches are then used as the input for the structure-level matching. Structural matching is based on the assumption that whenever two elements in the two graph models are found to be similar, similarity in their adjacent elements increases, using so-called ambiguous propagation. This process is repeated until the similarities of elements reach a fixpoint, and a collection of candidate matches is produced. With the aim of discovering a best match for each element, Similarity Flooding strictly returns a subset of 1-to-1 matches from the candidate matches.

- iMAP [DLD$^+$04] discovers complex semantic n-to-m attribute matches between two relational schemas using instance-level information. It consists of three main procedures that are applied in sequence: generating candidate matches, evaluating similarities and selecting final matches. The *generating* procedure uses a collection of searchers. Each searcher contains a manually specified formula and aims at quickly detecting a relatively small set of promising candidate matches for each attribute that satisfies its formula. As such, it is possible for a single attribute to participate in more than one candidate match. The searchers (formulae) currently supported by iMAP include, for example, *Text* (e.g., name = concat(first-name,last-name)), *Numeric* (e.g., list-price = price * (1 + tax-rate)), *Date* (e.g., birth-date = b-day/b-month/b-year) and *Unit conversion* (e.g., weigh-kg = 2.2 * net-weight-pounds). The *evaluating* procedure revises the similarity of each candidate match produced in the previous step by combining similarities from name-based and instance-level matchers. Finally, the *selecting* procedure searches for the best global match assignment among the candidate matches by considering domain constraints (for example, that each attribute can only take part in a single match).

- Dumas [BN05] matches attributes between two relational databases. Instead of comparing instances of individual attributes, this approach compares tuples of relational tables to derive the matches between attributes, using so-called horizontal string-based instance matching. Dumas makes intensive use of string comparison methods, including edit distance measures, token-based similarity measures and hybrid distance measures [CRF03].

Given source and target schemas, two collections of tuple strings are generated by concatenating attribute values in each tuple. Dumas first discovers the top k similar pairs of tuple strings, and then constructs a matrix for each tuple pair. The matrix stores similarity scores between each two attribute values that belong to the tuple pair. Finally, it combines the k matrixes together into a single matrix and calculates the overall average similarity scores to derive the attribute-level matches.

- LSD [DDH01] is an instance-level matching method that semi-automatically discovers the 1-to-1 matches between a mediated schema and local schemas in data integration scenarios, where source data is stored in XML files that conform to DTDs. It is a machine learning method that operates in two phases: the *training* phase asks users to manually specify matches in order to train the basic matchers using the instances of associated attributes; then the *matching* phase applies the internal classification rules obtained in the *training* phase for matchers to identify a collection of candidate matches. Similar to iMAP, domain constraints, such as a constraint on the number of matches for each element, are required to select a small set of matches from the candidate matches.

- Xu *et al.* [XE06] propose a method that exploits both schema-level and instance-level information to derive 1-to-1 and n-to-m matches. The method takes as input two schemas represented by their conceptual-model specifications [LEW00]. The authors propose a hybrid approach that applies the element-level and structure-level techniques successively. It derives similarities of individual elements by comparing their names (semantic-based matching) and data values (string-based matching). Furthermore, it applies manually specified formulae (e.g., concatenating and decomposing strings) to investigate the transformation between element instances, and identifies the n-to-m matches between elements that directly contain instances (e.g., attributes). During the structure-level matching step, this method matches elements only when their higher-level elements in the hierarchical graphs have been identified as being similar during the previous element-level matching process (top-down propagation). To derive the n-to-m matches between higher-level elements (i.e., elements that indirectly contain instances, such as tables in the relational model), it uses conceptual-model specific constraints to describe the internal relationship among the n or m elements. In the matching results returned by Xu *et al.*, a single element usually participates in several matches (i.e., 1-to-1 or n-to-m), thus requiring users to select their desired results.

2.5 View Generation

As we have mentioned in Section 1.2, views play significant roles in several application scenarios. For example, in data integration [HRO06, Len02], views are used to reformulate queries posed over a mediated schema into queries over local schemas; in data exchange [FHH$^+$09, FKMP03], they restructure and translate instances of a source schema into instances of a target schema; and in model management 2.0, schema operators (e.g., *Merge*, *Compose* and *Diff*) are provided [BM07, MBHR05, MAB07], where views serve as scripts that describe the relationships between two schemas and offer support functions for these operators. Views are usually generated by interpreting matches, as presented in Section 1.3. In this section, we briefly present some issues regarding views, including their definitions and a brief introduction to methods for (semi-)automatic view generation. We also illustrate how matches produced by schema matching techniques described in Sections 2.1 to 2.4 can be used to generate views.

2.5.1 View definition

A view is a virtual relation, whose content is expressed as a query over a (set of) base relation(s) or other views in the databases [Kot09]. In data integration, views explicitly specify the semantic relationships of elements between the global schema (or mediated schema) and the local schemas, so as to underpin the reformulation of a query over a global schema into a set of queries over the local schemas. So far, the research community has proposed three basic approaches for defining views in traditional data integration systems, called Global-As-View (GAV), Local-As-View (LAV), and more recently Global-and-Local-As-View (GLAV). Specifically,

- in the GAV approach [GMPQ$^+$97, ACPS96, Ull00], each element in the global schema is expressed as a view over the local schemas. The immediate benefit of GAV is that this kind of mappings explicitly specifies how to retrieve the source data in terms of a query posed over elements of the global schema. However, the GAV approach is mostly suited for scenarios where the local sources are comparatively stable, because adding a new local source to the current system or changing the schemas of already integrated sources requires modification of the global schema definition, and thus requires the redefinition of the associated views;

- in the LAV approach [DGL00, FW97], each local schema is expressed as a view over the global schema. As such, extending the system with a new source would not require the revision of the existing view definitions and would only require extending the system with a new view. On the other hand, answering queries using views is a relatively harder task [Hal01], because each view only describes partial information about the local sources in terms of the global schema. Thus, techniques for rewriting queries [BLR97] aim to find a query expression on the local schemas using a set of views that is equivalent to a given query over the global schema.

- dissatisfied with the disadvantages of both GAV and LAV, the GLAV approach [FLM99, HIST03] establishes the views by combining the expressive power of both GAV and LAV, while allowing for the flexibility of extending the integration system with a new local source. Basically, the GLAV approach defines views as a collection of assertions, in the form of two conjunctive queries (i.e., select-project-join queries) over the global and the local schemas, respectively. The conjunctive queries specify that elements of the global schema correspond to elements of the local schemas [Len02].

More recently, the research community has also dedicated effort to the languages and grammars that specify the view definitions, mostly using the GLAV approach. Examples are source-to-target tuple generating dependencies (tgds) [BV84] used by *Clio* [FHH+09], An *et al.* [ABMM07], *GeRoMe* [KQ+09, KQLJ07] and Papotti *et al.* [PT09], second-order tgds [FKPT04] developed by Fagin *et al.*, and a few self-defined mapping languages used in specific applications [PB08, ABBG09, MBHR05, YP04, ALM09].

2.5.2 Generating views

Since views are so important in several scenarios, some enterprise tools have been developed to address the problem of view generation, as described and evaluated in a recent survey [LN07]. However, most research proposals mentioned in Section 2.5.1 have only developed languages and approaches to defining rather than generating views [MH03]. In what follows, we will briefly introduce research prototypes that involve (semi-)automatic view generation. As can be observed, automatic view generation is not a simple problem, so most research prototypes that address this problem have to rely on complete schema definitions (e.g., referential integrity) or external resources (e.g., rich specification for the relationships of schemas) for help.

- The Clio project has been developed for ten years, and aims to address the problem of generating views (mappings) automatically as needed for data integration and data exchange [FHH$^+$09]. The Clio tool takes as input two schemas (e.g., relational and XML schemas) and the matches between the schemas (Clio authors call them value correspondences) to specify views. Over the last ten years, the Clio tool has gradually grown from a research prototype [FKMP03, Kol05] that discovered queries as GLAV [MHH00, PVM$^+$02] in the form of source-to-target *tuple generating dependencies* (tgd) [BV84] to an industrial tool that produces executable queries [HHH$^+$05]. However, as the input matches are inherently ambiguous [PVM$^+$02], they may have many interpretations. Several views that are consistent with the matches are produced, but not all of them have the correct meaning. As such, what Clio offers is an alternative to precise views: a collection of likely views is generated by enumerating different interpretations of the matches, which requires additional user feedback for the most appropriate views to be chosen. The Clio tool also makes extensive use of schema information (e.g., referential constraints as well as relational and nesting structures), in order to join or union schema elements, and as such to express views in the form of conjunctive queries required by GLAV. In the cases where complete schema information is absent (e.g., foreign keys are not defined), it is difficult for the Clio tool to generate views automatically.

- Model Management 2.0 [BM07] is the evolution of Model Management [BHP00, Ber03]. The aim of these approaches is to provide a generic set of operators for manipulating data models and mappings between them. Both versions of the approach comprise a wide range of operators, such as *Match* (i.e., identifying similar elements between two models), *Diff* (i.e., discovering differences between two models), *Merge* (i.e., merging two models into a third model using mappings between them), and *Compose* (i.e., combining two mappings into one mapping). The major difference between the two versions lies in the mapping representation used by the operators. In Model Management, mappings are the results of the *Match* operator, i.e., matches that relate elements between data models in an unspecified way [BM07]. However, as many application scenarios (e.g., data integration) manipulate data models that contain both metadata and instances, the simple matches are not enough to specify the relationships between instances, and as such views are proposed to replace matches with mappings in Model Management 2.0 [BM07]. Thus, a simple method for automatic view generation is proposed, which takes as input two schemas and the matches between them and produces view expressions in relational algebra [MBHR05] (e.g., $\pi_{att_1,att_2}(table_1 \bowtie table_2) = \pi_{att'_1,att'_2}(table'_1 \bowtie table'_2)$). In this approach, tables are joined on primary and foreign keys, and attributes associated by matches are taken as equivalent.

- Pottinger *et al.* [PB08] address the general problem of data integration by inferring a mediated schema from the local schemas and generating views between them. The method is built on the assumption that apart from the local schemas themselves, there exists a specification of their overlapping parts that can be used as input and has been presented in the form of conjunctive queries. For the purpose of capturing all the semantic information presented in the local schemas, the mediated schema is designed to contain all their elements, including both the overlapping parts and the source-specific elements. To construct the mediated schema, the method incrementally merges each local schema into the existing mediated schema by extending it with the source-specific elements and combining the overlapping elements. With the explicit and rich specification of local schema relationships in hand, this method generates GLAV between the mediated schema and the local schemas. Pottinger *et al.* demonstrate the effectiveness of the generated GLAV on query rewriting, which, however, requires a rich semantic specification of relationships among local sources, which is not always available.

Given matches between two schemas, Clio [FHH+09] and Model Management 2.0 [MBHR05] usually require a complete schema definition, especially information about primary keys and foreign keys, in order to generate views in the form of conjunctive queries. This is because tables can be joined into conjunctive queries only if paths between primary keys and foreign keys are given. Clio allows different views to be associated with a single element (e.g., a table in relational schemas), because it has to enumerate different interpretations of the input matches to generate the views. Therefore, Clio requires user effort to pick their desired views. Although Pottinger *et al.* [PB08] can automatically construct a mediated schema and generate GLAV between the mediated schema and local schemas, it relies on a strong assumption that the overlapping parts of the local schemas are specified in the form of conjunctive queries. The question arises as to whether or not the input conjunctive queries can be generated automatically.

Approaches to (semi-)automatic generation of views have been developed in the past few years, but they may not be suitable to be applied for dataspaces. As pointed out in Section 1.2.2, a dataspaces management system may start with providing low quality services but at low cost, which can be gradually improved as more effort is poured in. The setup of the dataspaces management system mostly emphasizes the automation of the process, and thus users are not expected to select views from a set of candidate views. Furthermore, dataspaces target at integrating syntactically diverse data sources (e.g., relational databases, ontologies, or personal information in the form of text documents) and thereby consider explicit referential constraints optional to schema definitions. Therefore, relying on primary keys and foreign keys to identify join paths between elements is not always applicable in dataspaces. The above disadvantages displayed in the existing methods for generating views are due to the ambiguous information carried by the matches. Additional research on inferring relationships between two schemas that offer more semantics than the matches is necessary and important.

2.6 Summary and Conclusions

In this chapter, we presented the general research context of inferring schematic correspondences, including techniques for identifying matches and generating views. We defined the schema matching operator in Section 2.1, followed by a comprehensive description of schema-level and instance-level techniques in Sections 2.2 and 2.3, respectively. We brought various matching techniques together through presenting the state-of-the-art schema matching systems in Section 2.4. A general observation of these systems is that one-to-one elements are associated mostly because they have similar names or instances rather than because there is firm evidence that they represent the same concept. It is common that two elements representing different concepts are matched because of similar names. For example, considering RDB1 and RDB2 in Figure 1.1, overseas_customer.name and cust_account.account_name could be matched, thus giving rise to incorrect views (e.g., View *2* in Section 1.3). In addition, matches produced by the existing schema matching systems cannot explicitly indicate how to join two elements (e.g., tables in a relational database) inside a schema, and as such cannot offer enough information for generating the conjunctive queries in GLAV. Clio and Model Management 2.0, therefore, assume that referential constraints are available explicitly in schema definitions to compensate for the inadequacy of matches, which is not desirable for dataspaces, as discussed in Section 2.5.

The research described in this thesis aims to infer schematic correspondences between source and target schemas based on matches, in order to provide sufficient information for the automatic generation of views [MBPF09]. However, the ability of matches to represent schematic correspondences and the technical gap between identifying matches and inferring schematic correspondences have not been appropriately established. Hence, we propose a two-step research to achieve the goal of this thesis: i) to evaluate the effectiveness of existing schema matching systems on diagnosing schematic heterogeneities (Chapters 3 and 4); and ii) based on lessons learnt from the evaluation, to devise an approach to inferring schematic correspondence (Chapter 5) and to evaluate its effectiveness (Chapter 6).

Chapter 3

MatchBench

In this chapter, we present MatchBench, a benchmark that evaluates the effectiveness of schema matching proposals in terms of their ability to diagnose the schematic heterogeneities presented in Section 1.1. We expect to understand the underlying techniques of the schema matching proposals, and identify a collection of existing matchers that produce matches suitable for inferring schematic correspondences. MatchBench employs a generator to create a wide range of synthetic scenarios with well defined characteristics in terms of of schematic heterogeneities, against which schema matching systems are evaluated. MatchBench also provides positive and negative experiments for testing each type of schematic heterogeneity, where the positive experiment assesses the effectiveness of the matchers when a heterogeneity is present, while the negative experiment measures them when the heterogeneity is absent.

This chapter is structured as follows. Section 3.1 illustrates the related work to MatchBench, followed by Section 3.2 that presents an overview of the MatchBench method. Sections 3.3 and 3.4 introduce the collection of synthetic scenarios and the description of experiments designed for diagnosing schematic heterogeneities, respectively. Section 3.5 summarizes the whole chapter.

3.1 Related Work

Before moving on to describe MatchBench in detail, we first discuss related work, with the aim of showing the novelty and contributions made with our benchmark. In contrast to Chapter 2 that merely provides a general research background for the whole thesis, which reviews schema matching techniques and representative matching systems, this section discusses previous work on evaluating schema matching systems. In particular, we revisit established criteria for assessing such systems, the evaluation test cases that have been synthetically generated, and existing benchmarks for testing schema matching and mapping approaches in Sections 3.1.1, 3.1.2 and 3.1.3, respectively.

3.1.1 Experimental evaluation of schema matching

In a review of schema matching evaluation, Do *et al.* [DMR02] summarized the main criteria for assessing schema matching tools by surveying published reports of experimental evaluations. The criteria include:

- *input*, which refers to the input data employed by the matching approaches, including *schema type*, *number of schemas*, *schema scale* and *auxiliary information*, as introduced in Section 2.1.2. Do *et al.* also proposed *schema similarity* in this criterion which is the ratio between the number of correct matches identified manually and the number of elements in both input schemas.

- *output*, which indicates the information provided about the matched elements. Do *et al.* break down this criterion further as *element representation* that describes match types (i.e., matches between element nodes or between element paths) and *match cardinality* as presented in Section 2.1.2.

- *quality measures*, which are used to evaluate the system performance. The metrics discussed by Do *et al.* mostly follow the canonical evaluation model of *Information Retrieval* [BYRN99], i.e., *Precision*, *Recall* and *F-measure*, but also include *Overall* introduced in Similarity Flooding [MGMR02]. In particular, the performance of the matching approaches can be measured by comparing their results with the ground truth, i.e., the correct matches, thus allowing to determine true positives (TP), i.e., matches correctly identified; false positives (FP), i.e., matches incorrectly identified; and false negatives (FN), i.e., matches incorrectly missed; true negatives (TN), i.e., false matches that are correctly discarded. Given the cardinalities of the above sets (i.e., $|TP|$, $|FP|$, $|FN|$ and $|TN|$), *Precision*, *Recall*, *F-measure* and *Overall* are defined as follows:

- *Precision* $= \frac{|TP|}{|TP|+|FP|}$ specifies the fraction of correct matches among all detected matches;
- *Recall* $= \frac{|TP|}{|TP|+|FN|}$ specifies the fraction of correct matches among all detectable matches;
- *F-measure* $= 2 * \frac{Precision*Recall}{Precision+Recall}$ is the harmonic mean of *Precision* and *Recall*;
- *Overall* $= \frac{|TP|-|FP|}{|TP|+|FN|} = Recall * (2 - \frac{1}{Precision})$ estimates the labor savings obtained by using an automatic matcher, i.e., effort needed for adding the false negatives and removing the false positives.

- *effort*, which assesses the human involvement in the process of running matchers, including *pre-match effort* required before running the matchers (e.g., training these matchers, configuring their parameters, and specifying auxiliary information), and *post-match effort* to add the false negatives and remove the false positives, as measured by *Overall.*

Instead of suggesting a set of standard criteria for evaluating matching systems, Do *et al.* [DMR02] only offer a summary of existing evaluation criteria, because most systems have chosen specific criteria to illustrate their effectiveness in particular contexts. Do *et al.* also explicitly stated that the diversity of the evaluation criteria has been problematic: "*While there have been some evaluations, the overall effectiveness of currently available automatic schema matching systems is largely unclear. This is because the evaluations were conducted in diverse ways making it difficult to assess the effectiveness of each single system, let alone to compare their effectiveness.*" [DMR02]. This motivates MatchBench, in demonstrating that there is a need to develop a benchmark for evaluating and comparing matching systems in more general contexts and specifying the matching tasks (i.e., input), the expected results (i.e., output) and the quality measures.

Recently, some other metrics have been proposed, in addition to these introduced by Do *et al.* to complement the *quality measures* introduced above:

- *response time*, which refers to the running time required by the matchers to complete a matching task [ASS09, CHT05, DKS^{+}08, BEFF06, WT06, KN08, DR07].

- *matching accuracy*, which is defined as the percentage of source/target attributes that are matched correctly [DDH01, DLD+04].

While designing MatchBench, we decided to use some of the evaluation criteria introduced by Do *et al.*, because they are well-known in the research community, thus being understandable and justifiable. In particular, we choose pairs of small scale relational schemas that represent one or more types of schematic heterogeneities as the *input*; we expect that 1-to-1, n-to-1 (1-to-n) or n-to-m individual elements are matched as the *output*; we measure the effectiveness of schema matching systems by reporting *Precision*, *Recall* and *F-measure* but not *Overall*.

Ontology Alignment Evaluation Initiative (**OAEI**) [EFH+09] is the most comprehensive evaluation activity on ontology matching, and runs an annual competition. This involves different types of test case that address different aspects of ontology matching. Every year, the OAEI organizers first publish a few initial test cases online[1], which allows potential participants to send observations, correct mistakes or suggest other test cases. After all test cases have been finalized, participants are required to run their matching systems automatically on these cases and report matching results to the organizers. The participants are also requested to use the same configuration throughout the competition, which, however, could be the one that is able to achieve the best results. Finally, the organizers will evaluate the results of the matching approaches used by the participants and provide comparisons and conclusions. Specifically, we categorize the OAEI event according to the evaluation criteria introduced previously:

- the *input* released by the OAEI organizers consists of ontology files of varying scale ranging from small to large. The input ontologies are categorized into tracks, each of which represents a specific matching context, including a *comparison* track that provides several synthetic tests and introduces variations (e.g., by changing attribute names or removing attributes) into two identical schemas, *expressive ontologies* and *directories and thesauri* tracks that offer real world examples in special domains.

- the *output* produced by ontology matching consists of ontology alignments. Most of them are 1-to-1 matches between ontology elements.

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

- the *quality measures* used by the OAEI organizers include *Precision*, *Recall*, *F-measure*. *Response time* is reported where required, especially in the cases of matching large scale ontologies.

MatchBench shares some common properties with the OAEI, for example, the synthetic test cases and the use of *Precision*, *Recall*, *F-measure* as the quality measures. However, the synthetic test cases provided by the OAEI seem *ad hoc*; MatchBench, in contrast, explicitly follows a systematic procedure to inject different types of schematic heterogeneities. Instead of simply assessing the quality performance, MatchBench focuses on the extent to which systematically introduced variations affect the ability of the system to identify specific schematic correspondences, and thus different experiments have been developed to address this goal. The OAEI has not addressed this overall requirement in its synthetic tests.

3.1.2 Generation of test cases for schema matching and mapping

eTuner [SLDR05, LSDR07] is an approach to systematically setting control parameters in schema matching systems, supported by a collection of automatically generated test cases. In essence, eTuner considers this setting process as a search problem that seeks effective configurations for parameter values under certain circumstances. The configuration space of a matching system is defined to be different combinations of component matchers that potentially require candidate values for various control parameters. In order to tune a matching system, eTuner first generates a collection of synthetic test cases for which the ground truth is known. It then applies the system on such test cases, compares the results to the ground truth and reports the F-measures. The configuration with which the matching system has achieved the highest F-measure is identified as an effective parameter setting. The test schemas over which eTuner evaluates the systems are generated by applying a number of rules for introducing perturbations into existing schemas. These perturbations overlap with those described in Section 3.3, but differ in the following respects:

- rather than following an established classification of schematic heterogeneities, they represent a limited enumeration of possible matching scenarios;

- their ground truth contains only 1-to-1 matches, while MatchBench offers scenarios whose ground truth are collections of 1-to-1 and n-to-m matches; and

- they do not offer negative scenarios to evaluate whether systems do not match elements that should not be matched.

STBenchmark [ATV08] is a benchmark for comparing visual interactive mapping construction systems that aim at assisting an expert in generating a precise specification of mappings between two schemas with less effort. STBenchmark creates a basic suite of mapping scenarios that are the result of a careful analysis of various data integration applications. Instead of exhaustively enumerating all possible mapping scenarios, the authors only intend to present common cases that exhibit wide industrial relevance. In essence, the MatchBench scenarios described in Section 3.3, where different types of variations are injected into an initial pair of schemas, overlap with STBenchmark scenarios but exhibit more variety, as follows:

- STBenchmark provides a basic tool for injecting different types of variation into initial schemas in order to produce pairwise task schemas. However, both the selection of types of variation and the generation of evaluation scenarios rest on the shoulders of users. On the other hand, MatchBench does not emphasize its contribution to injection techniques, but focuses on offering scenarios that represent different types of schematic heterogeneity, thus allowing users to investigate the influence of the combined heterogeneities captured by the scenarios.

- STBenchmark mainly focuses on injecting structural perturbations on the initial schemas, e.g., horizontal and vertical partitioning, nesting and flattening; MatchBench, on the other hand, not only considers the structural perturbations but also atomic changes, e.g., changing table and attribute names.

3.1.3 Benchmarks for schema matching and mapping

XBenchMatch [DBH07] has been developed as a benchmark in the context of XML schema matching. It provides a software tool for computing properties of given scenarios, applying matching systems and deriving matching results.

XBenchMatch chooses four sets of real world XML schemas as matching tasks, each of which represents a matching scenario (e.g., a set of schemas that describe persons, or a set of schemas that cover university courses). Given the ground truth, this tool compares them with the matching results of the evaluated systems and reports *Precision*, *Recall* and *F-measure*. Although both XBenchMatch and MatchBench contribute to evaluating schema matching systems, their major differences lie in the following:

- The scenarios used by XBenchMatch to compare the matching systems are selected from real world examples. As such, their ability to explore a systematic collection of representative issues, or at least the common cases as covered by STBenchmark, is unknown. In contrast, MatchBench systematically explores its chosen context of schematic heterogeneities, and concentrates on this diagnostic assessment.
- XBenchMatch focuses more on building a benchmark and only reports the matching quality of tested systems (e.g., *F-measure*) without analytical conclusions, whereas MatchBench develops a benchmark in a different context, and compares and analyzes the effectiveness of the tested systems in its specific context.

STBenchmark [ATV08] has been discussed above in terms of its test case generation capability. It has also been designed for assessing the human effort required to implement a mapping through the visual interface of interactive mapping systems, such as Clio [HHH$^+$05], Microsoft BizTalk[2] and Stylus Studio[3], rather than measuring the quality of specified mappings. As such, STBenchmark is complementary to MatchBench. Insights from MatchBench may inform the development of helper components for interactive mapping tools that suggest to users what mappings may be most appropriate in a given setting.

3.2 Overview of MatchBench

The aim of this thesis, i.e., to infer schematic correspondences between two schemas, requires a good understanding of existing schema matching proposals, as the correspondences will be inferred from matches. Therefore, we have designed

[2]http://www.microsoft.com/biztalk/en/us/default.aspx
[3]http://www.stylusstudio.com

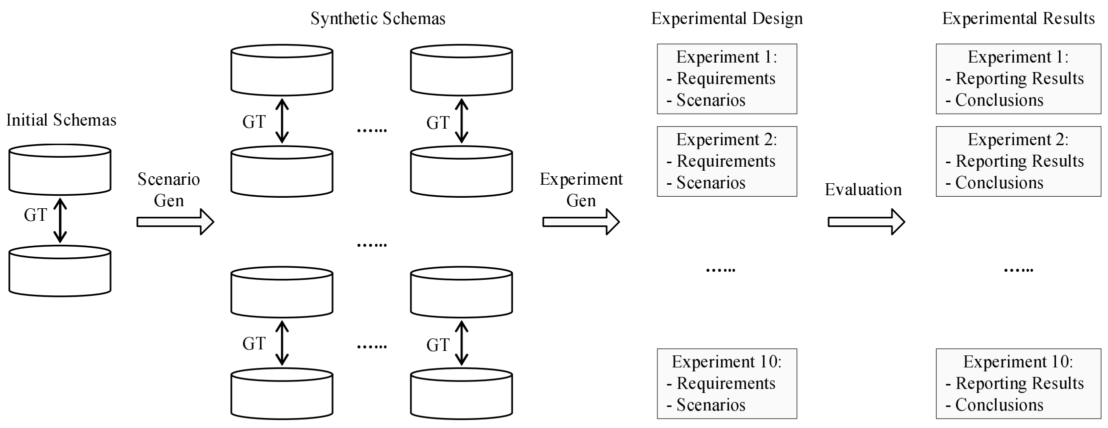


Figure 3.1: MatchBench Work Flow.

and implemented MatchBench to obtain knowledge about the latter. The hypothesis behind MatchBench is that the effectiveness of matching algorithms can be evaluated in terms of their ability to diagnose the schematic heterogeneities described in Section 1.1. The objectives of MatchBench are:

1. to identify the extent to which different matching systems can diagnose the heterogeneities of interest;
2. to establish under what circumstances specific systems struggle to diagnose which kind of heterogeneities;
3. to support the identification of complementary matchers building on the results of 1 and 2.

Meeting such objectives requires systematic investigation into how matching systems perform given a collection of scenarios that exhibit the heterogeneities of interest. As such, MatchBench uses the following operators to meet this requirement, as illustrated in Figure 3.1:

- *ScenarioGen* is the operator that generates a collection of new scenarios by systematically injecting different types of schematic heterogeneities into a pair of initial schemas. Thus, the strength of available evidence to support the diagnosis of schematic heterogeneities in different scenarios varies in a controlled manner. In essence, each new scenario consists of two schemas that represent one or more types of schematic heterogeneity and the ground truth that defines equivalent parts. Details about the injection process and the generated scenarios is presented in Section 3.3.

- The *ExperimentGen* operator constructs 10 pre-designed experiments that assess matching tools in the context of each particular schematic heterogeneity. Each experiment provides a set of scenarios that represent a heterogeneity, and states the requirements that the matching tools are expected to achieve in order to diagnose the heterogeneity, as presented in detail in Section 3.4.

- The *Evaluation* operator runs selected schema matching tools on the generated scenarios, investigates whether their matching results meet the requirement to diagnose specific schematic heterogeneities, and reports the *precision*, *recall* and *F-measure*.

3.3 MatchBench Scenarios

This section discusses the wide range of synthetic scenarios that represent schematic heterogeneities described in Section 1.1.2. The initial pair of databases, into which the schematic heterogeneities are systematically injected, is presented in Section 3.3.1. MatchBench generates both *positive* and *negative* scenarios, as described in Sections 3.3.2 to 3.3.5, whose scenario numbers are summarized in Table 3.1. Totally, MatchBench generates 336 scenarios.

Scenario Space	Number of Scenarios
Positive scenario space for one-to-one entity correspondences	207
Negative scenario space for one-to-one entity correspondences	24
Positive scenario space for many-to-one attribute correspondences	9
Positive scenario space for many-to-many entity correspondences	96
Total	336

Table 3.1: The number of scenarios generated by MatchBench.

In the *positive* scenarios, the starting point is an initial pair of databases that have a single table in common, into which schematic heterogeneities are systematically introduced. They are intended to be representative of the sorts of heterogeneity that exist in practice between independently developed databases representing the same domain. In the *negative* scenarios, the starting point is that there exists no common information between the initial pair of databases (by removing the common table), into which similarities are systematically introduced, giving rise to scenarios where tables should not be matched, but there are some similarities between them. These negative scenarios have been designed for the purpose of evaluating the extent to which matching algorithms can accommodate certain similarities but there is little other evidence that matches should be postulated.

3.3.1 Initial schemas

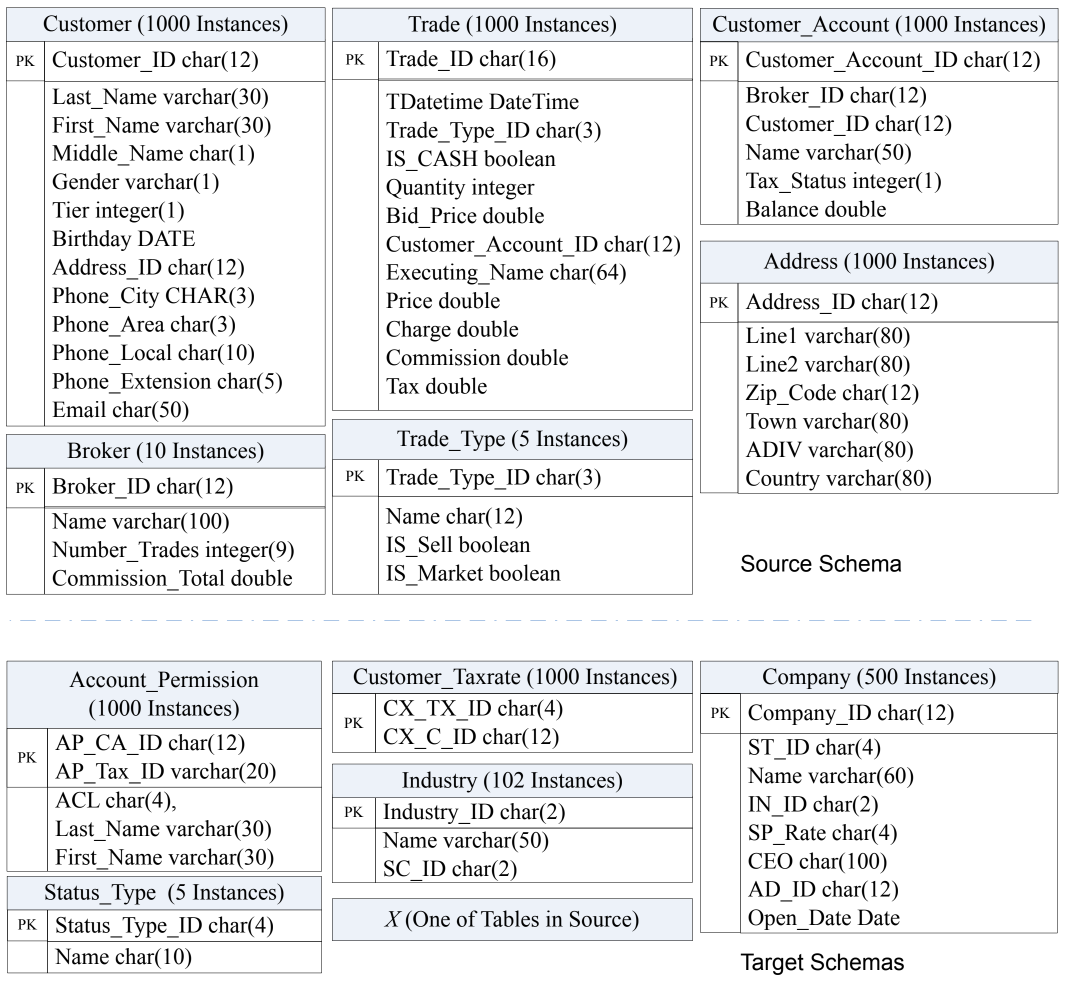


Figure 3.2: The *source* and *target* databases used as a basis for scenario generation.

Figure 3.2 describes the data source, which has been derived from TPC-E[4], into which schematic heterogeneities are injected to generate the scenarios in MatchBench. Instances for the tables have been produced using the TPC-E

[4]http://www.tpc.org/tpce/

generator. From the 33 tables of TPC-E, the 11 tables shown in Figure 3.2 are generated by removing a small number of foreign keys or by joining two tables into a single one, combined with some renaming of tables and attributes.

In essence, these 11 tables, with the given cardinalities, have been used because they exhibit considerable variety in the numbers, types and domains of attributes, and because the extents are small enough to allow memory-intensive matching algorithms to execute on standard desktop hardware.

3.3.2 Positive scenarios for one-to-one entity correspondences

In positive scenarios, the initial *target* database is extended with a single table from the *source*, and thus the two databases have a pair of common tables to which schematic heterogeneities are injected. Figure 3.3 describes the space of positive scenarios where one-to-one heterogeneities are introduced into the pair of identical tables between the source and target databases. In the implementation of MatchBench, we chose three source tables (particularly *Customer*, *Customer_Account* and *Trade*) as the target corresponding table in Figure 3.2, because they exhibit comparatively more variety in the numbers and types of attributes than the rest.

Figure 3.3 (with examples in Figure 3.4 where schematic heterogeneities are injected into an example pair of equivalent entities) shows 8 scenario sets, each of which comprises a set of scenarios and represents one or more schematic heterogeneities between a pair of equivalent entities. We use the solid line boxes to represent scenario sets and arrows to represent the systematic introduction of heterogeneities into the scenario sets. Each scenario set and its specified scenarios manifest examples of the heterogeneities named in the corresponding solid line box, the definitions of which are provided below. For example, *Scenario Set 1* represents the starting point for the introduction of the heterogeneities, and the arrow leading to *Scenario Set 5* indicates that it has been derived from *Scenario Set 1* through the changing of entity[5] names.

In what follows, where names are described as the *same*, they are identical, and where they are described as *similar*, their strings overlap (in the implementation of MatchBench, we form a new name by removing a small set of characters from

[5]In MatchBench, we use the terms defined in Section 1.1.2, and thus table, entity and entity types are taken to be synonyms.

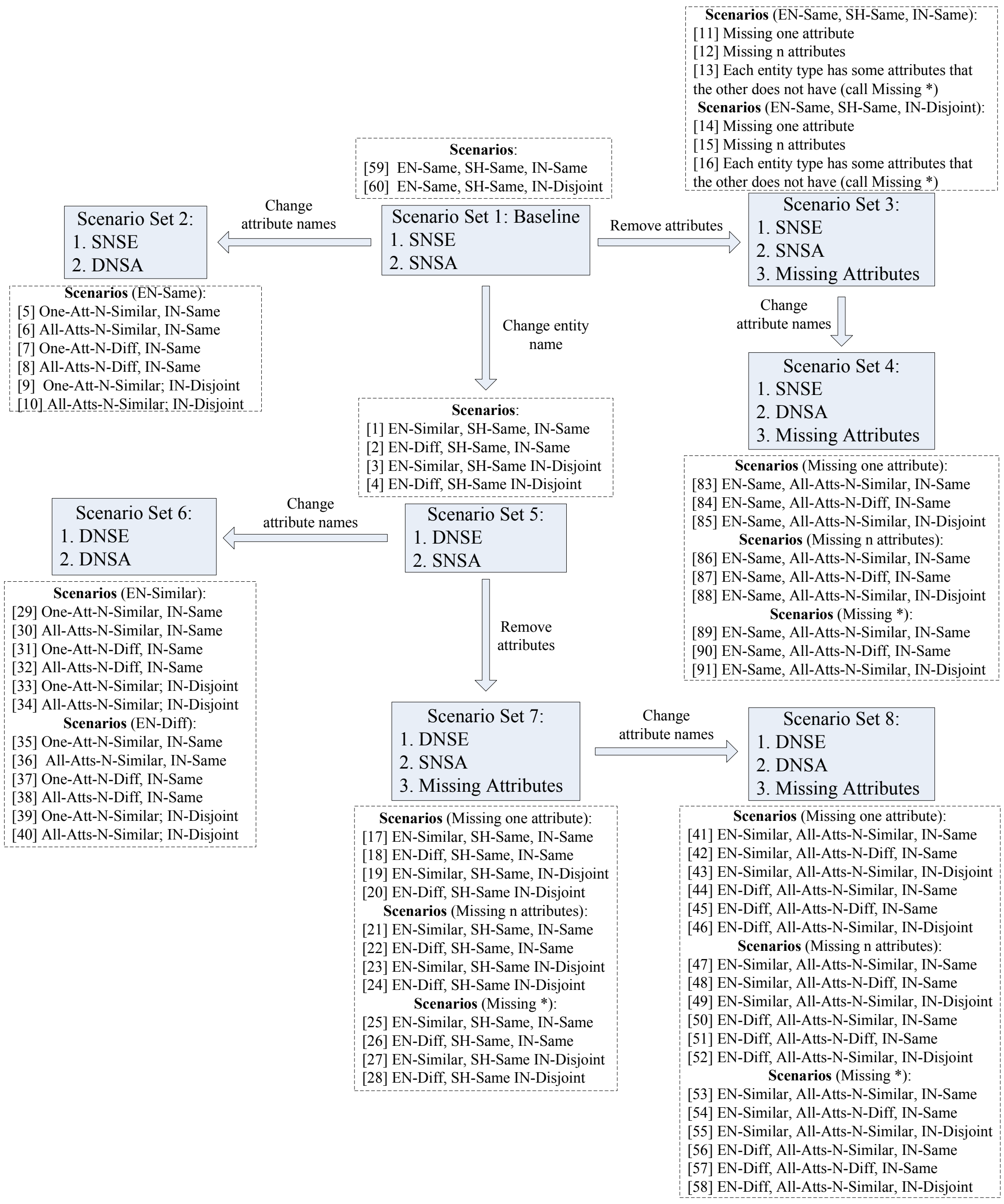

Scenarios (EN-Same, SH-Same, IN-Same):
[11] Missing one attribute
[12] Missing n attributes
[13] Each entity type has some attributes that the other does not have (call Missing *)
Scenarios (EN-Same, SH-Same, IN-Disjoint):
[14] Missing one attribute
[15] Missing n attributes
[16] Each entity type has some attributes that the other does not have (call Missing *)
Scenarios:
[59] EN-Same, SH-Same, IN-Same
[60] EN-Same, SH-Same, IN-Disjoint
Scenario Set 2:
1. SNSE
2. DNSA
Change attribute names
Scenario Set 1: Baseline
1. SNSE
2. SNSA
Remove attributes
Scenario Set 3:
1. SNSE
2. SNSA
3. Missing Attributes
Scenarios (EN-Same):
[5] One-Att-N-Similar, IN-Same
[6] All-Atts-N-Similar, IN-Same
[7] One-Att-N-Diff, IN-Same
[8] All-Atts-N-Diff, IN-Same
[9] One-Att-N-Similar; IN-Disjoint
[10] All-Atts-N-Similar; IN-Disjoint
Change entity name
Change attribute names
Scenario Set 4:
1. SNSE
2. DNSA
3. Missing Attributes
Scenarios:
[1] EN-Similar, SH-Same, IN-Same
[2] EN-Diff, SH-Same, IN-Same
[3] EN-Similar, SH-Same IN-Disjoint
[4] EN-Diff, SH-Same IN-Disjoint
Scenarios (Missing one attribute):
[83] EN-Same, All-Atts-N-Similar, IN-Same
[84] EN-Same, All-Atts-N-Diff, IN-Same
[85] EN-Same, All-Atts-N-Similar, IN-Disjoint
Scenarios (Missing n attributes):
[86] EN-Same, All-Atts-N-Similar, IN-Same
[87] EN-Same, All-Atts-N-Diff, IN-Same
[88] EN-Same, All-Atts-N-Similar, IN-Disjoint
Scenarios (Missing *):
[89] EN-Same, All-Atts-N-Similar, IN-Same
[90] EN-Same, All-Atts-N-Diff, IN-Same
[91] EN-Same, All-Atts-N-Similar, IN-Disjoint
Scenario Set 6:
1. DNSE
2. DNSA
Change attribute names
Scenario Set 5:
1. DNSE
2. SNSA
Scenarios (EN-Similar):
[29] One-Att-N-Similar, IN-Same
[30] All-Atts-N-Similar, IN-Same
[31] One-Att-N-Diff, IN-Same
[32] All-Atts-N-Diff, IN-Same
[33] One-Att-N-Similar; IN-Disjoint
[34] All-Atts-N-Similar; IN-Disjoint
Scenarios (EN-Diff):
[35] One-Att-N-Similar, IN-Same
[36] All-Atts-N-Similar, IN-Same
[37] One-Att-N-Diff, IN-Same
[38] All-Atts-N-Diff, IN-Same
[39] One-Att-N-Similar; IN-Disjoint
[40] All-Atts-N-Similar; IN-Disjoint
Remove attributes
Scenario Set 7:
1. DNSE
2. SNSA
3. Missing Attributes
Change attribute names
Scenario Set 8:
1. DNSE
2. DNSA
3. Missing Attributes
Scenarios (Missing one attribute):
[17] EN-Similar, SH-Same, IN-Same
[18] EN-Diff, SH-Same, IN-Same
[19] EN-Similar, SH-Same, IN-Disjoint
[20] EN-Diff, SH-Same IN-Disjoint
Scenarios (Missing n attributes):
[21] EN-Similar, SH-Same, IN-Same
[22] EN-Diff, SH-Same, IN-Same
[23] EN-Similar, SH-Same IN-Disjoint
[24] EN-Diff, SH-Same IN-Disjoint
Scenarios (Missing *):
[25] EN-Similar, SH-Same, IN-Same
[26] EN-Diff, SH-Same, IN-Same
[27] EN-Similar, SH-Same IN-Disjoint
[28] EN-Diff, SH-Same IN-Disjoint
Scenarios (Missing one attribute):
[41] EN-Similar, All-Atts-N-Similar, IN-Same
[42] EN-Similar, All-Atts-N-Diff, IN-Same
[43] EN-Similar, All-Atts-N-Similar, IN-Disjoint
[44] EN-Diff, All-Atts-N-Similar, IN-Same
[45] EN-Diff, All-Atts-N-Diff, IN-Same
[46] EN-Diff, All-Atts-N-Similar, IN-Disjoint
Scenarios (Missing n attributes):
[47] EN-Similar, All-Atts-N-Similar, IN-Same
[48] EN-Similar, All-Atts-N-Diff, IN-Same
[49] EN-Similar, All-Atts-N-Similar, IN-Disjoint
[50] EN-Diff, All-Atts-N-Similar, IN-Same
[51] EN-Diff, All-Atts-N-Diff, IN-Same
[52] EN-Diff, All-Atts-N-Similar, IN-Disjoint
Scenarios (Missing *):
[53] EN-Similar, All-Atts-N-Similar, IN-Same
[54] EN-Similar, All-Atts-N-Diff, IN-Same
[55] EN-Similar, All-Atts-N-Similar, IN-Disjoint
[56] EN-Diff, All-Atts-N-Similar, IN-Same
[57] EN-Diff, All-Atts-N-Diff, IN-Same
[58] EN-Diff, All-Atts-N-Similar, IN-Disjoint

Figure 3.3: Positive scenario space for one-to-one entity correspondences.

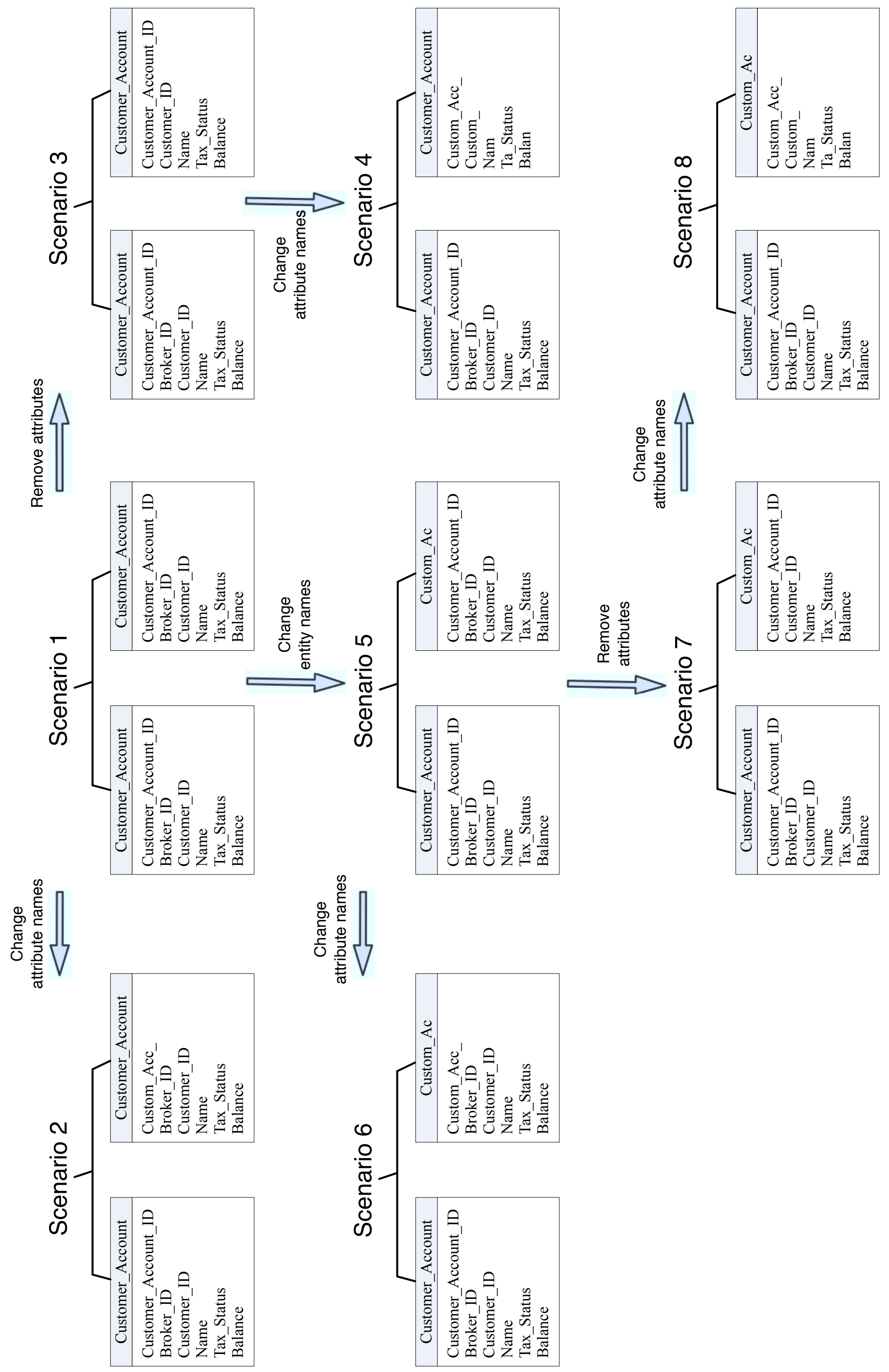


Figure 3.4: Examples for the positive scenario space for one-to-one entity correspondences.

an original name string); neither of these properties hold for *different* names. The terms below are used in Figure 3.3 to describe the properties of scenario sets, following the terminology introduced in Section 1.1.2:

- *SNSE* (Same Name for Same Entity types): the names of equivalent entity types are the same.
- *DNSE* (Different Names for Same Entity types): the names of equivalent entity types are similar or different.
- *SNSA* (Same Name for Same Attribute): the names of equivalent attributes are the same.
- *DNSA* (Different Names for Same Attribute): the names of one or more equivalent attributes are similar or different.
- *Missing Attributes*: one or more attributes are missing.

The following terms describe the properties of specified scenarios, each of which has been labelled with a unique number:

- *EN-Same*, *EN-Similar* or *EN-Diff*: the names of equivalent entity types are the same, similar or different, respectively.
- *SH-Same*: the schemas of equivalent entity types are the same, i.e., all pairs of equivalent attributes have the same name.
- *IN-Same* or *IN-Disjoint*: the extents of equivalent entity types contain either the *same instances* (SI) or *disjoint instances* (DI). *Disjoint instances* are generated by partitioning a single collection of instances, and thus have common attribute values but no identical tuples.
- *One-Att-N-Similar* or *One-Att-N-Diff*: a single pair of equivalent attributes has similar or different names, and the remaining pairs have the same name.
- *All-Atts-N-Similar* or *All-Atts-N-Diff*: all pairs of equivalent attributes have similar or different names.

3.3.3 Negative scenarios for one-to-one entity correspondences

The space of negative scenarios for one-to-one entity types is described in Figure 3.5. In the implementation of MatchBench, we choses three pairs of entity types that represent different real world information into which similarities are systematically injected (particularly between source tables *Customer*, *Customer_Account* and *Trade* and the target table *Company*, respectively). The following terms are used to describe the properties of the scenario sets:

- *DNDE* (Different Names for Different Entities): all pairs of entity types are different, and so are their names.
- *SNDE* (Same Name for Different Entities): two entity types are different but have the same name.
- *DNDA* (Different Names for Different Attributes): all (or all except one) pairs of attributes are different and so are their names.
- *SNSA* (Same Name for Same Attribute): two equivalent attributes have the same name.
- *DNSA* (Different Names for the Same Attributes): two equivalent attributes have different names.
- *SNDA* (Same Name for Different Attributes): two different attributes have the same name.

The terms that are used to describe the properties of the scenarios in each scenario set are presented as follows:

- *EN-Same* or *EN-Diff*: different entity types have the same or different names.
- *All-Atts-N-Diff-Inst-Diff*: all attributes of different entity types have different names and different instances.
- *(n-1)-Atts-N-Diff-Inst-Diff*: all but one attributes of different entity types have different names and different instances.

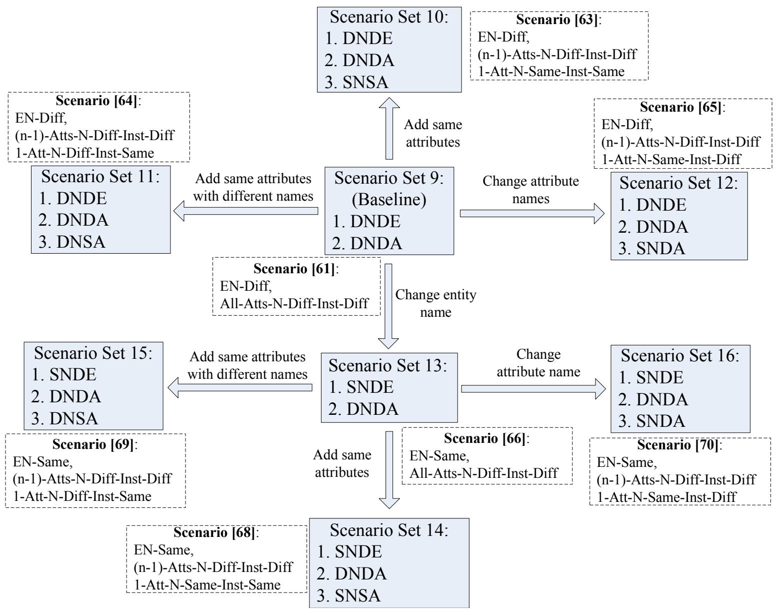


Figure 3.5: Negative scenario space for one-to-one entity correspondences.

- *1-Att-N-Diff-Inst-Same*: a single pair of attributes of different entity types have different names but the same instances.
- *1-Att-N-Same-Inst-Diff*: a single pair of attributes of different entity types have the same name but different instances.
- *1-Att-N-Same-Inst-Same*: a single pair of attributes of different entity types have the same name and instances.

3.3.4 Positive scenarios for many-to-one attribute correspondences

In Figure 3.6, the space of attribute many-to-one correspondences is described, where a set of attributes and a single attribute that belong to equivalent entity types represent the same real world information. The following properties describe the scenario sets:

- *Attribute Many-to-One Correspondence types*: three different types of attribute many-to-one correspondences are considered:
  - *Type 1*, numeric operation:
    $(price + charge + commission) \times (1 + tax) = price$

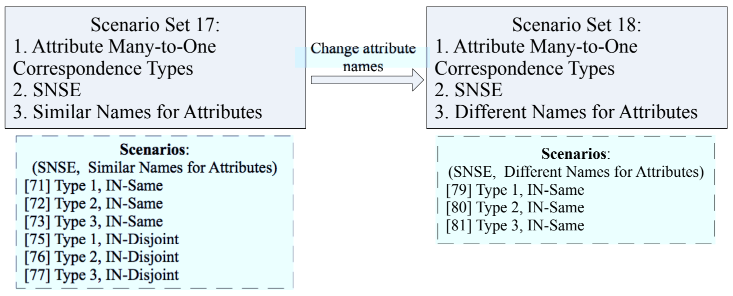


Figure 3.6: Positive scenarios for many-to-one attribute correspondences.

- *Type 2*, string concatenation:
  Concat (*first_name*, *middle_name*, *last_name*) = *name*
- *Type 3*, numeric concatenation:
  Concat (*phone_city*, *phone_area*, *phone_local*, *phone_extension*) = *phone*

- *Similar Names for Attributes*, which indicates that some or all of the *many* attributes have similar names to the *one* corresponding attribute.
- *Different Names for Attributes*, which indicates that all of the *many* attributes have different names from the *one* corresponding attribute.

Similar to Figure 3.3, extents of equivalent entity types are generated so as to give rise to SI and DI cases for scenario set 17. Scenario set 18 contains SI cases, but not DI, in order to retain a certain level of similarity between attributes.

3.3.5 Positive scenarios for many-to-many entity correspondences

The initial *target* database is extended with a single table from the *source*, i.e., *Customer*, *Customer_Account* and *Trade*, respectively. In this space, as described in Figure 3.7, two sets of entity types (called the source and the target) represent the same real world information. The following properties characterize the scenario sets:

- *Entity Many-to-Many Correspondence types:* three different types of many-to-many correspondences are considered; for each type, the property of the

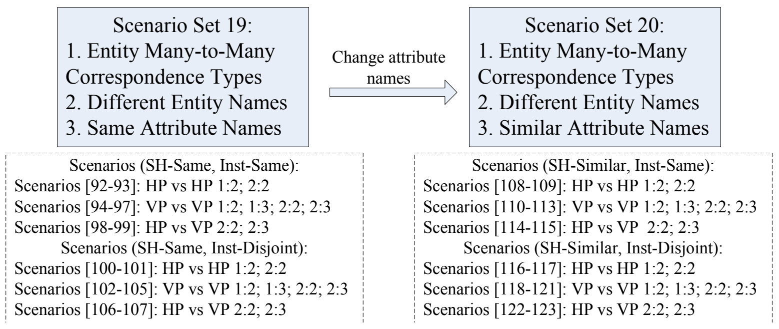

Scenario Set 19:
1. Entity Many-to-Many Correspondence Types
2. Different Entity Names
3. Same Attribute Names
Change attribute names
Scenario Set 20:
1. Entity Many-to-Many Correspondence Types
2. Different Entity Names
3. Similar Attribute Names
Scenarios (SH-Same, Inst-Same):
Scenarios [92-93]: HP vs HP 1:2; 2:2
Scenarios [94-97]: VP vs VP 1:2; 1:3; 2:2; 2:3
Scenarios [98-99]: HP vs VP 2:2; 2:3
Scenarios (SH-Same, Inst-Disjoint):
Scenarios [100-101]: HP vs HP 1:2; 2:2
Scenarios [102-105]: VP vs VP 1:2; 1:3; 2:2; 2:3
Scenarios [106-107]: HP vs VP 2:2; 2:3
Scenarios (SH-Similar, Inst-Same):
Scenarios [108-109]: HP vs HP 1:2; 2:2
Scenarios [110-113]: VP vs VP 1:2; 1:3; 2:2; 2:3
Scenarios [114-115]: HP vs VP 2:2; 2:3
Scenarios (SH-Similar, Inst-Disjoint):
Scenarios [116-117]: HP vs HP 1:2; 2:2
Scenarios [118-121]: VP vs VP 1:2; 1:3; 2:2; 2:3
Scenarios [122-123]: HP vs VP 2:2; 2:3

Figure 3.7: Positive scenarios for many-to-many entity correspondences.

form *n:m* below represents the cardinalities of the source and target sets, respectively. The types are:

  – *HP vs HP* (1:2; 2:2), where the source and target sets are related by horizontally partitioning the original entity type.
  – *VP vs VP* (1:2; 1:3; 2:2; 2:3), where the source and target sets are related by vertically partitioning the original entity type.
  – *HP vs VP* (2:2; 2:3), where the source and target sets are related by horizontal and vertical partitioning the original entity type, respectively.

- *Different Entity Names*, which indicates that the names of the entity types in the source and target sets are different.
- *Same Attribute Names*, which indicates that equivalent attributes have the same names.
- *Similar Attribute Names*, which indicates that equivalent attributes have similar names.

3.4 Experiments

3.4.1 Effectiveness measures

We decided to follow some of the evaluation criteria introduced by Do *et al.*, specifically the *Precision*, *Recall* and *F-measure* presented in Section 3.1, to assess the effectiveness of different matching systems for diagnosing schematic heterogeneities. This is because they are well-known in the research community, thus being understandable and justifiable. Therefore, other alternative measures, such as, *Accuracy* are not applied in MatchBench.

3.4.2 Experiment design

Building on the scenarios described above, MatchBench offers a collection of experiments that can be used to explore the effectiveness of different schema matching systems in diagnosing the presence of schematic heterogeneities. In most of the experiments described below, the *F-measure* is reported, but where this is useful for interpretation, the *precision* and *recall* may be reported separately.

MatchBench includes five positive and five negative experiments. Usually, a positive experiment evaluates the extent to which the matching systems are able to diagnose the presence of a schematic heterogeneity. A negative experiment, on the other hand, tests the effectiveness of the systems in not reporting such a schematic heterogeneity. Hence, a higher *F-measure* reported in the positive experiments indicates better performance of the systems. Negative experiments in turn report *1 - F-measure* so that a lower value indicates better performance.

Specifically, Experiments *1* and *2*, *3* and *4*, *5* and *6*, *7* and *8*, and *9* and *10* are pairwise positive and negative experiments, and evaluate *DNSE*, *DNSA*, *missing attributes*, *many-to-one attributes* and *many-to-many entities* heterogeneities. The experiments are defined as follows.

Experiment 1: *Identifying when the same entity occurs in positive scenarios.* This experiment involves *Scenario Sets 1* to *8* in Figure 3.3, and reports on the ability of the matchers to meet two requirements:

- *Requirement R1:* Equivalent entity types are matched, where the ground truth is the set of pairwise entity correspondences between equivalent entity types. Thus,
  - *true positives* are the identified matches between the equivalent entity types;
  - *false positives* are the incorrectly identified matches between different entity types;
  - *false negatives* are the matches between equivalent entity types that are

incorrectly missing.

- *Requirement R2:* Equivalent attributes are matched, where the ground truth is the collection of pairwise attribute correspondences between equivalent attributes. Thus,
  - *true positives* are the identified matches between the equivalent attributes;
  - *false positives* are the incorrectly identified matches between different attributes;
  - *false negatives* are the matches between equivalent attributes that are incorrectly missing.

Experiment 2: *Identifying when the same entity occurs in negative scenarios.* This experiment involves *Scenario Sets 9* to *16* in Figure 3.5, and reports on the extent to which the matchers meet the following requirements where no correspondences exist:

- *Requirement R1:* Different entity types are not matched, where the ground truth is that there are no pairwise entity correspondences between different entity types. Thus,
  - *true positives* are the different entity types that are not associated with matches;
  - *false positives* are the matches incorrectly associated with the different entity types;
  - *false negatives* are the different entity types that are associated with matches.

- *Requirement R2:* Different attributes are not matched, where the ground truth is that there are no pairwise attribute correspondences between attributes of the different entity types. Thus,
  - *true positives* are attributes of the different entity types that are not associated with matches;
  - *false positives* are the matches incorrectly associated with attributes of the different entity types;
  - *false negatives* are attributes of the different entity types that are associated with matches.

The results reported for Experiments *1* and *2* provide the evidence necessary to establish the presence of both the *same name for different entities* and *different* *names for equivalent entities* heterogeneities of Kim *et al.* [KS91], and also play a role in the detection of subsequent correspondences.

Experiments 3-8 are carried out only when the evaluated systems are able to identify that the two entity types are equivalent; as such only correspondences that associate attributes of equivalent entity types are measured.

Experiment 3: *Identifying where different names have been given to equivalent attributes in positive scenarios.* This experiment involves *Scenario Sets 2, 4, 6* and *8* in Figure 3.3, where the ground truth is the collection of pairwise attribute correspondences between equivalent attributes that have similar or different names. Thus,
- *true positives* are the identified matches between equivalent attributes that have similar or different names;
- *false positives* are the identified incorrect matches that associate equivalent attributes that have similar or different names with different attributes;
- *false negatives* are the matches that are incorrectly missing between equivalent attributes that have similar or different names.

Experiment 4: *Identifying where different names have been given to equivalent attributes in negative scenarios.* This experiment involves scenarios from *Scenario Sets 3, 4, 7* and *8* in Figure 3.3, in which attributes have been removed from the corresponding entity types in both the *source* and *target* databases, thus resulting in equivalent entity types with different attributes. Note that the negative scenarios in this experiment (and in Experiments *6*, *8* and *10*) make use of properties of the data sets created for testing positive scenarios – distinct data sets have only been created specifically for negative scenarios where this is necessary. The ground truth of the experiment is that there is no pairwise attribute correspondence between different attributes. Thus,
- *true positives* are attributes that do not have corresponding attributes and are not associated with matches, and as such are correctly identified as different attributes;
- *false positives* are attributes that have corresponding attributes but are not associated with matches, and as such are incorrectly identified as different attributes;
- *false negatives* are attributes that do not have corresponding attributes but are associated with matches, and as such are not identified as different attributes.

Experiment 5: *Identifying missing attributes in positive scenarios.* This experiment involves *Scenario Sets 3, 4, 7* and *8* in Figure 3.3, where some attributes have been removed. The ground truth is that there are no correspondences associated with the missing attributes (defined as attributes whose counterparts have been removed). Thus,
- *true positives* are the missing attributes that are correctly identified, i.e., which are not matched to any other attributes;
- *false positives* are the non-missing attributes that are not associated with matches, and as such are incorrectly identified as missing attributes;
- *false negatives* are the missing attributes that are incorrectly matched to other attributes.

Experiment 6: *Identifying missing attributes in negative scenarios.* This experiment involves *Scenario Sets 1, 2, 5* and *6* in Figure 3.3, where every attribute has a counterpart. The ground truth is the collection of pairwise attribute correspondences between each attribute and some corresponding attribute. Thus,
- *true positives* are the non-missing attributes that are associated with matches;
- *false positives* are the non-missing attributes that are not associated with matches, and as such are incorrectly identified as missing attributes;
- *false negatives* are the non-missing attributes that are not associated with matches.

Experiment 7: *Identifying many-to-one attribute correspondences in positive scenarios.* This experiment involves *Scenario Sets 17* and *18* in Figure 3.6, where each element in the ground truth is a collection of attribute correspondences between each attribute in the set and the single attribute. Thus,
- *true positive* is the identified collection of matches between each attribute in the set and the single attribute (i.e., these attributes are not associated with other attributes);
- *false positives* are the identified collections of many-to-one or one-to-many attribute matches that are not the ground truth (i.e., the collection of attribute matches that is contained by, contains, overlaps or is disjoint with the ground truth collection of matches);
- *false negative* is the collection of matches between each attribute in the set and the single attribute, which is not identified.

Experiment 8: *Identifying many-to-one attribute correspondences in negative scenarios.* This experiment involves *Scenario Sets 1* to *8* in Figure 3.3, where

there are one-to-one but no many-to-one attribute correspondences. The ground truth is that there are no many-to-one or one-to-many correspondences associated with each attribute. Thus,
- *true positives* are the attributes that do not participate in more than one match;
- *false positives* are the collections of matches that associate many-to-one or one-to-many attributes;
- *false negatives* are the attributes that participate in more than one match.

Experiment 9: *Identifying many-to-many entity correspondences in positive scenarios.* This experiment involves *Scenario Sets 19* and *20* in Figure 3.7, and explores the following requirements in the context of the categories described in Section 3.3.5:

- *Requirement R1*: Each entity type in the source set should be matched to all entity types in the target set. The ground truth is the collection of pairwise entity correspondences between each entity type in the source set and all entity types in the target set. Thus,
  - *true positive* is the identified collection of matches between each two entities in the source and target sets (i.e., these entities are not associated with other entities);
  - *false positives* are the identified collections of many-to-one, one-to-many, or many-to-many entity matches that are not the ground truth (i.e., the collection of entity matches that is contained by, contains, overlaps or is disjoint with the ground truth collection of matches);
  - *false negative* is the collection of matches between each two entities in the source and target sets, which is not identified.

  The following two requirements are investigated only when the evaluated systems are able to meet *R1*. As such, to diagnose them, only correspondences that associate attributes of corresponding entity types in the source and target sets are measured.

- *Requirement R2*: Primary key attributes in each entity type in the source set should be matched to primary key attributes in all entity types in the target set. The ground truth is the collection of pairwise attribute correspondences between primary key attributes in each entity type in the source set and primary key attributes in all entity types in the target set. The specified *true positives*, *false positives* and *false negatives* are similar to these of

Requirement R1, thus being omitted.

- *Requirement R3*: Partitions in the source schema are matched against partitions in the target schema, with a view to identifying specific types of many-to-many correspondences. For each type, the ground truth is the collection of pairwise attribute correspondences between attributes as described below:
  - *Horizontal Partitioning vs Horizontal Partitioning:* Each non-key attribute in each entity type in the source (target) set should be matched to a single non-key attribute in every entity type in the target (source) set.
  - *Vertical Partitioning vs Vertical Partitioning:* Each non-key attribute in each entity type in the source (target) set should be matched to a single non-key attribute in an entity type in the target (source) set.
  - *Horizontal Partitioning vs Vertical Partitioning:* Each non-key attribute in each entity type in the source set should be matched to a single non-key attribute in an entity type in the target set; but each non-key attribute in each entity type in the target set should be matched to a single non-key attribute in each entity type in the source set.

  The specified *true positives*, *false positives* and *false negatives* are similar to these of *Requirement R1*, thus being omitted.

Experiment 10: *Identifying many-to-many entity correspondences in negative scenarios.* This experiment involves *Scenario Sets 1* to *8* in Figure 3.3, and explores the following requirements:

- *Requirement R1*: Each entity type in the source should not be matched to more than one entity type in the target. The ground truth is that there are no many-to-many entity correspondences associated with each entity type in both source and target. Thus,
  - *true positives* are entity types that do not participate in more than one match;
  - *false positives* are the collections of entity matches that associate many-to-many entity types;
  - *false negatives* are entity types that participate in more than one match.

- *Requirement R2*: Each attribute in each entity type in the source set should not be matched to an attribute in each entity type in the target set, and thus will not be identified as many-to-many attributes required in diagnosing many-to-many entity conflicts. The ground truth is that there are no many-to-many attribute correspondences for each attribute in both source and target set. The specified *true positives*, *false positives* and *false negatives* are similar to these of *Requirement R1*, thus being omitted.

Requirement R2 is investigated only when the evaluated systems are unable to meet *Requirement R1*. If that is the case, many-to-many entity correspondences are identified between two sets of entity types (called the source and target) by the evaluated systems. Thus, to diagnose *R2*, only correspondences that associate attributes of matched entity types in the above two sets are measured.

3.5 Summary and Conclusions

This chapter has presented MatchBench, a benchmark that investigates the effectiveness of schema matching proposals in the context of schematic heterogeneities introduced earlier in Section 1.1. MatchBench offers a wide range of synthetic scenarios with well defined characteristics of schematic heterogeneities, which distinguishes from eTuner [SLDR05, LSDR07] and STBenchMark [ATV08] as discussed in Section 3.1.2. Adopting the evaluation criteria summarized by Do *et al.* [DMR02], MatchBench applies the synthetic scenarios as the input for matching systems, and compares their output with the requirements for identifying each type of schematic heterogeneities. In particular, MatchBench makes the following contributions:

- it offers a wide range of synthetic scenarios that are systematically generated from a pair of schemas and represent different types of schematic heterogeneities; and
- it defines positive and negative experiments to assess schema matching tools in terms of their ability to diagnose the schematic heterogeneities, where a positive experiment investigates the tools when a heterogeneity is present, and a negative experiment assesses the tools when the heterogeneity is absent.

Chapter 4

Application of MatchBench

In this chapter, we discuss the application of MatchBench to three well-known schema matching platforms, namely COMA++ [DR07, DR02, EM07], Rondo [MRB03] and OpenII [SMH$^+$10]. We chose to evaluate these approaches because they are among the best known schema matching platforms, and are publicly available. We describe the three matching platforms and their parameters used by MatchBench in Section 4.1. We compare their effectiveness in diagnosing schematic heterogeneities in Section 4.2, followed by conclusions in Section 4.3.

4.1 Matching Systems

We discuss configurations required to run the three schema matching platforms in this section. In principle, we follow the advice of the authors on how to configure them, for example, by employing the settings suggested in the published papers or in direct interactions. In addition, we do what we can to help the three platforms to perform well, e.g., by plugging an instance-level matcher into Rondo and OpenII that are only supplied with the schema-level matchers. We describe the configurations of COMA++, Rondo and OpenII in Sections 4.1.1, 4.1.2, and 4.1.3, respectively, followed by a summary in Section 4.1.4.

4.1.1 COMA++ configuration

As introduced in Section 2.4, COMA++ [DR07] is a schema matching platform that provides a combination of schema-based and instance-based matching supported by a library of matchers. It extends COMA [DR02] to include more schema-level matchers and matching strategies, and complements it with instance-level matching [EM07]. The matching behaviour exhibited by COMA++ can be configured as follows:

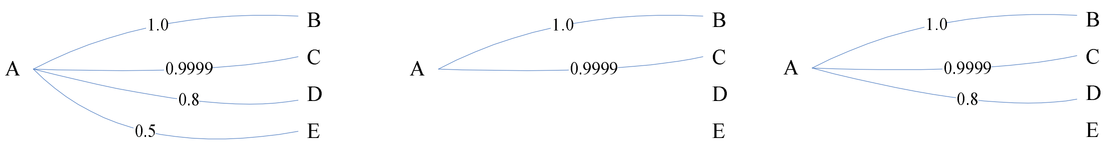
- *Choose a matching strategy.* Strategies can be chosen from (i) *AllContext*, which matches paths from the root to nodes in hierarchical data sets; (ii) *NoContext*, which only considers matching of individual nodes; and (iii) *FilteredContext*, which seeks to match paths of nodes only when the nodes are identified as similar. As our schemas consist of flat tables, this aspect of configuration has no impact in principle, although *AllContext* is used in practice as it is the only strategy that enables the combination of schema and instance-based matching in the COMA++ implementation.

- *Choose the matchers.* The following matcher combinations were selected from the COMA++ library because they have been demonstrated to be effective in published experimental evaluations [DR07, DR02, EM07, Do06]. The following schema-level matchers are used: (i) the string matchers *Name*, which determines the similarity of element names, and *NamePath*, which takes the context (table names) of elements into account; (ii) the structure-level matchers *Leaves*, which derives similarity of attributes from a composite matcher *NameType* (which takes element names and data types into account) and propagates the similarity from attributes to entity types, and *Parents*, which obtains similarity of entity types from the *Leaves* matcher and propagates the similarity from entity types to attributes without combining information of entity types (e.g., their name similarity). At the instance level, the *Content-based* matcher is used, which performs pairwise string comparison of instance values to match attributes, and propagates similarity from attributes to entity types.

- *Choose the combination parameters.* Combining correspondences from different matchers includes four steps: (i) *aggregation*, which determines a combined similarity for two elements, configured as *Average* that assigns even weights to the results of schema and instance matching; (ii) *direction*, which ranks elements of a schema (e.g., S_1) based on similarity of matches, set up as *Both* that matches S_1 to S_2 and S_2 to S_1; (iii) *selection*, which chooses matches above a specific confidence level by applying a *Threshold+MaxDelta*, as will be discussed below; (iv) *combined similarity*, which decides a combined similarity between sets of matched elements, given the value of *Average*. The experimental study carried out by COMA++ authors [DR07, Do06] identified *Average*, *Both*, *Threshold+MaxDelta* and *Average* as the most effective strategies for *aggregation*, *direction*, *selection* and *combined similarity*, respectively, for cases where no previous matches are reused.

Before moving on, we need to introduce more detail of the *Threshold+MaxDelta* method used for *selection*, as COMA++ results are fairly sensitive to its configured values, i.e., the *threshold* and *delta* values below. Specifically, given a collection of candidate matches (i.e., matches produced by combining results of various matchers and to be selected as final results), *Threshold+MaxDelta* returns the intersection of matches chosen by the *Threshold* method and by the *MaxDelta* method:

- the *Threshold* method selects matches whose similarities pass a *threshold* value set by the user;
- the *MaxDelta* method selects matches associated with an element whose similarities reside in a tolerance range specified by $[max - max \times delta, max]$, where *max* denotes the maximum similarity of the matches associated with the element, and *delta* is a parameter to be configured.



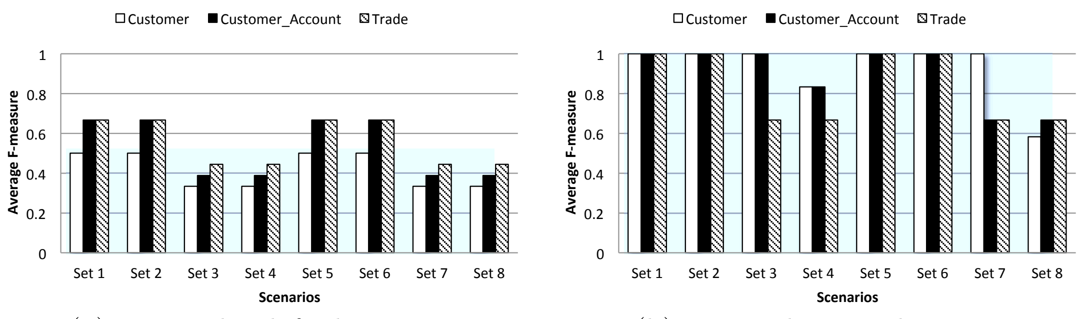
(a) Candidate matches (b) Matches selected by (0.1, 0.01) (c) Matches selected by (0.1, 0.3)

Figure 4.1: Example for the *Threshold+MaxDelta* method.

For example, assume that there are four candidate matches between attribute *A* and attributes *B*, *C*, *D* and *E*, as shown in Figure 4.1(a). Figure 4.1(b) shows the matches chosen by *Threshold+MaxDelta* where (*threshold*, *delta*) are set as (0.1, 0.01): the *Threshold* method selects all four matches, and the *MaxDelta* method chooses the matches between *A* and *B* and between *A* and *C*, as the tolerance range is specified as [1.0–1.0×0.01, 1.0], which is [0.99, 1.0]. If (*threshold*, *delta*) are set as (0.1, 0.3), more matches are chosen, as shown in Figure 4.1(c):

the *Threshold* method still selects all four matches, but the *MaxDelta* method specifies the tolerance range as [1.0–1.0×0.3, 1.0], which is [0.7, 1.0], and as such matches between attribute A and attributes B, C and D will be returned.

The COMA++ authors suggested that setting (*threshold*, *delta*) to (0.5, 0.008) yields the best average quality (i.e., using F-measure and Overall, as introduced in Section 3.1.1) in the conducted experiments [Do06]. In Ontology Alignment Evaluation Initiative (OAEI) 2006 [EMS+06], COMA++ authors applied a (*threshold*, *delta*) of (0.13, 0.0001) [MER06]. It has proven difficult to configure (*threshold*, *delta*), which can have a significant impact on the performance of COMA++ results, as different parameter values for threshold and delta have been used in various publications by the authors. These differences indicate that the most effective values are context-specific. Following instructions of COMA++ authors [Do06]: "*With the flexibility to construct and configure matchers and match strategies, we have been able to quickly implement and test different match algorithms in various settings. This in turn allows us to identify the strategies and configurations with high and stable quality for our default match operation.*", we decided to employ the default setting of (*threshold*, *delta*) that is automatically configured as (0.1, 0.01) in the COMA++ tool we obtained from the authors for the evaluation using MatchBench.



(a) Using the default setting (b) Using the tuned setting

Figure 4.2: Comparison of COMA++ results for different settings of (*threshold*, *delta*): F-measure.

Even though during the following evaluations we employed the default setting for *threshold* and *delta*, we did an extra experiment to tune the two parameters on MatchBench scenarios, and have obtained values of (0.42, 0.33). Figure 4.2 shows the difference between COMA++ results in Experiment *1* for Requirement *R1* (i.e., identifying equivalent entity types) in the *same instances* scenarios by applying the default setting and by applying the tuned setting, respectively.

The reason why COMA++ reported higher F-measures in the tuned setting than in the default setting is that the *threshold* of *0.42* in the tuned setting excludes most false positives between different entities, whereas the *threshold* of *0.1* used by the default setting tends to return matches between entities as long as the similarity passes *0.1*. This indicates again that setting parameters is rather context-specific.

4.1.2 Rondo configuration

Rondo [MRB03] is a model management platform that provides various operators including, such as schema matching, merging, composition. For schema matching, Rondo uses an algorithm called Similarity Flooding [MGMR02], which consists of the following steps:

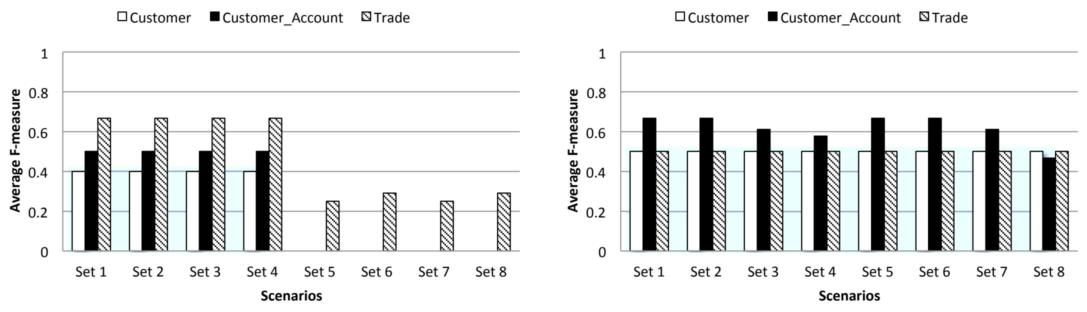
1. Build graph-based representations for schemas of the two data sources.

2. Produce initial matches using an *NGram* matcher that compares name strings of elements between the two data sources.

3. Run the similarity flooding algorithm: based on the assumption that whenever any two elements in the two graph models are found to be similar, the similarity of their adjacent elements increases, iteratively propagate similarities of elements in the matches derived from *Step 2* to their adjacent elements until a fixpoint is reached.

4. Select a subset of good matches from the candidate matches obtained in *Step 3*, whereby only one-to-one correspondences are produced.

For the evaluation of Similarity Flooding using MatchBench, the initial matches are produced by combining results of the *NGram* matcher and an instance matcher (i.e., the *Content-based* matcher of COMA++), rather than using the *NGram* matcher alone, which acts only on schema level data. This approach enables Similarity Flooding to make use of instance-level information in initial matches, which turns out to be important for identifying schematic correspondences. Figure 4.3 compares Requirement *R1* (i.e., identifying equivalent entity types) results of Rondo in the same instances scenarios of Experiment *1*, whose initial matches are produced without and with the instance matcher. As can be observed, Rondo



(a) Without the instance matcher (b) With the instance matcher

Figure 4.3: Comparison of Rondo results without and with the instance matcher: F-measure.

performs considerably worse in the absence of evidence from the instance matcher, especially when equivalent entities have similar or different names (Sets *5* to *8* in Figure 4.3(a) and (b)).

4.1.3 OpenII Configuration

OpenII [SMH$^+$10] is a suite of open-source tools designed for offering collaborative assistance (which means that users are required to contribute to the tasks) for information integration tasks, such as schema matching and data exchange. OpenII employs a hybrid matcher called Harmony [SMM$^+$09] that averages similarities of matches generated by the following matchers: *EditDistance*, which compares the edit distance of name strings of two elements, *Documentation*, which compares words found in the names and the documentation of two elements, and *Exact*, which identifies elements of source and target schemas that have exactly the same hierarchical path from the root. Harmony also provides the *Mapping* matcher, which reuses the previous results. We have not used *Mapping* for the evaluation with MatchBench, as we do not consider previous matches during the evaluation.

As an interactive matching tool, Harmony returns all candidate matches, so that the OpenII GUI can allow the user to slide a threshold bar while visually observing which matches pass different thresholds. The OpenII GUI also suggests a top match for each element (i.e., the match with the greatest similarity score), which is kept as long as it is the top match for either of its associated elements[1].

[1]OpenII developers also suggested that applying a threshold to remove the low score matches could improve the precision of the results. However, a suitable threshold that can be applied

As there are a large number of scenarios in MatchBench, selecting a threshold manually for each of them is not a practical proposition. Thus, we decided to follow the recommendation of the OpenII authors, i.e., use the top matches associated with each element. In addition, as Harmony only works at the schema-level, we combine it with the *Content-based* matcher of COMA++, to provide the same basis in terms of initial matches as COMA++ and Rondo.

4.1.4 Comparison of Configurations

In this section, we compare configurations chosen for the three matching platforms (see Table 4.1). In general, all three platforms combine the results of an instance-level matcher (i.e., the *Content-based* matcher of COMA++) with the results of their own schema-level matchers using average. Specifically, to generate candidate matches in COMA++ and OpenII, their individual schema-level matchers are combined using equal weights, and those results are in turn combined again with equal weights with the results of the instance-level matcher. Rondo propagates similarity of initial matches that are produced by averaging the results of the NGram and the *Content-based* matchers to generate the candidate matches, as stated in Section 4.1.2.

	Schema-level matchers	Instance-level matchers	Combination	Selection
COMA++	Name NamePath Parents Leaves	Content-based	Average	Threshold+MaxDelta, where (threshold, delta) = (0.1, 0.01)
Rondo	NGram	Content-based	Average	1-to-1 top matches
OpenII	EditDistance Documentation Exact	Content-based	Average	1-to-1 or n-to-m top matches

Table 4.1: Configurations of the three matching platforms.

In addition to collecting evidence for candidate matches differently, the three platforms also differ in the way candidate matches are selected. By introducing a tolerance range, *Threshold+MaxDelta* employed by COMA++ selects a few of the top matches, decided by the *delta* value, for each element that passes a *threshold*, thus identifying what could be seen as n-to-m matches. Rondo yields a single top match for each element only and thus returns one-to-one matches. OpenII, though is similar to Rondo in choosing a single top match for each element, allows

in MatchBench is hard to obtain from the past experience of OpenII developers. As such we decided to not use it in MatchBench.

a match to be kept as long as it is the top match for either of its associated elements, and as such could also identify n-to-m matches rather than just 1-to-1 matches.

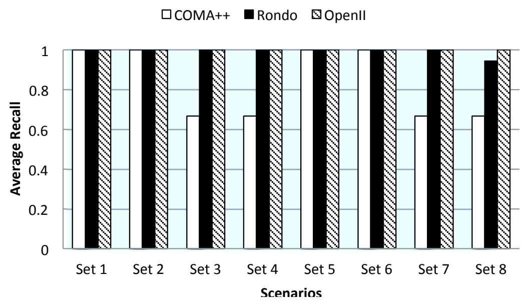
4.2 Effectiveness comparison

4.2.1 Experiment 1: Identifying when the same entity occurs in positive scenarios

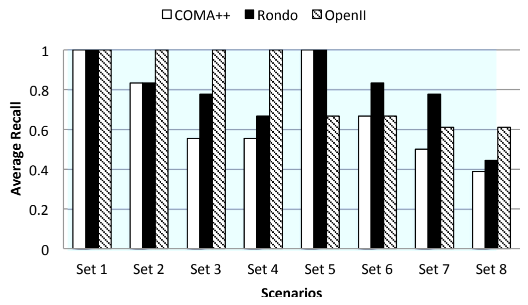
The *recall* and *F-measure* of this experiment are presented in Figures 4.4 and 4.5. In principle, we observe whether equivalent entity types and equivalent attributes can be matched using the *recall*, whereas the *F-measure* is used to report the overall performance of the three platforms and also reflects the incorrect matches between different entity types or different attributes. The following can be observed:

1. All three platforms have been generally successful at matching equivalent entity types and equivalent attributes when they have the same instances, but have been less successful for disjoint instances (Figure 4.4(a) and (b) for entity types, and (c) and (d) for attributes). OpenII has outperformed COMA++ and Rondo in terms of equivalent entity types and equivalent attributes alone (the *recall* reported in OpenII in Figure 4.4 is higher than the *recall* reported in COMA++ and Rondo in most cases, even though the *F-measure* reported is rather low in Figure 4.5).

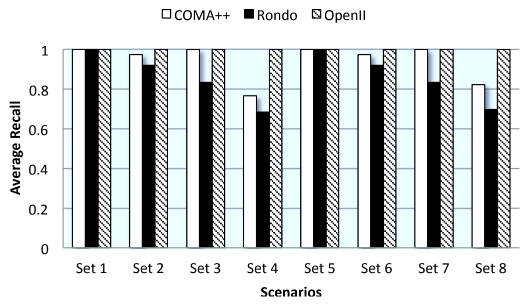
2. The importance of instance information for the three platforms is reflected in the fact that equivalent entity types that have the same instances can always be matched by Rondo and OpenII, irrespective of changes to their names and their attributes (Figure 4.4(a)). This is because matches between equivalent entity types that have same instances are usually their highest matches, and thus can be chosen. COMA++ has performed slightly worse than the other two platforms on matching equivalent entity types that have the same instances where some of their attributes are removed (Sets *3*, *4*, *7* and *8* in Figure 4.4(a)), because COMA++ also matches entity types to attributes. When similarity of equivalent entity types reduces, entities sometimes have higher similarity to attributes than to entities.



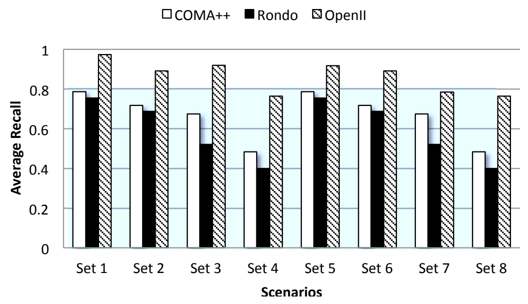
(a) Expt 1: R1, same instances



(b) Expt 1: R1, disjoint instances

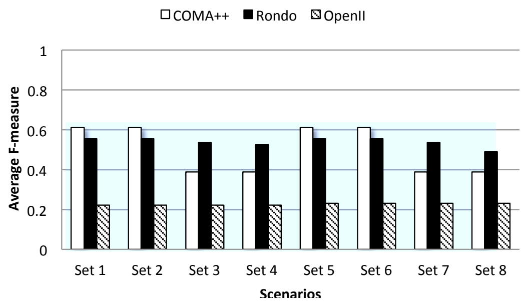


(c) Expt 1: R2, same instances

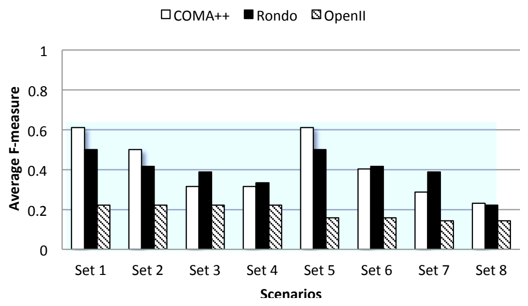


(d) Expt 1: R2, disjoint instances

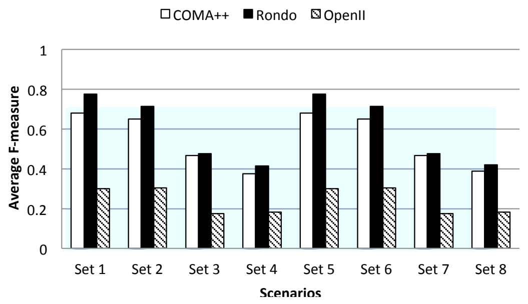
Figure 4.4: Experiment 1 for COMA++, Rondo and OpenII: Recall.



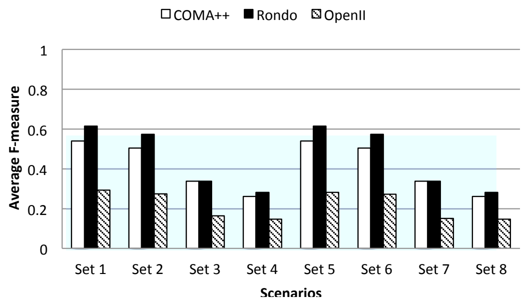
(a) Expt 1: R1, same instances



(b) Expt 1: R1, disjoint instances



(c) Expt 1: R2, same instances



(d) Expt 1: R2, disjoint instances

Figure 4.5: Results of Experiment 1 for COMA++, Rondo and OpenII: Average F-measure.

3. A significant number of false positives between different entity types and between different attributes have been generated by all platforms (given the high *recall* shown in Figure 4.4, the *F-measures* reported in Figure 4.5 are fairly low). This is due to the selection strategies these platforms employ: for COMA++, the *MaxDelta* method always chooses a few of the top matches associated with an element, even though the scores of these matches may be fairly low due to the low threshold of 0.1, which accepts a large number of matches with fairly low scores; Rondo returns one-to-one matches only, and does so by selecting a best match for each element, regardless of its similarity score; and OpenII keeps a match as long as it is the top match for either of its associated elements irrespective of the match scores, resulting in a large number of incorrect matches, which makes it perform worst among the three platforms.

4. Changing the names of equivalent entity types into similar or different influences OpenII slightly on matching equivalent attributes that have disjoint instances, but has no impact on COMA++ and Rondo (Figure 4.4(c) and (d)). This indicates that the *Parents* and *NamePath* matchers COMA++ employs and the Similarity Flooding algorithm used in Rondo do not propagate similarity of entity names into their attributes.

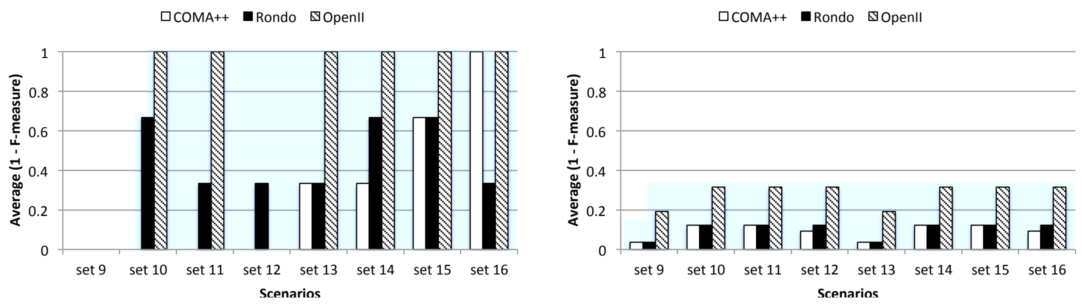
5. Given similar attribute names and the introduction of missing attributes, OpenII has been more robust than COMA++ and Rondo on matching equivalent attributes (Sets *2* to *4* and *6* to *8* in Figure 4.4(c) and (d)). The above changes to attributes tend to affect Rondo more than COMA++, because similarity of any two elements can be propagated to their neighbours (e.g., siblings) by similarity flooding.

In the following experiments (for COMA++, Rondo and OpenII), results are often better in the same instances case, and we comment on this in the text only where there is some additional point to be made.

As shown in Figure 4.5(c) and (d), results of COMA++, Rondo and OpenII between Scenario Sets *1* and *5*, *2* and *6*, *3* and *7*, and *4* and *8*, where the only difference is that entity names are changed, have shown no differences or slight differences. Thus, there is no need to repeat this general lesson in the text describing Experiments *3* to *6* and *8*, where results in Scenario Sets *5* to *8* are not reported.

4.2.2 Experiment 2: Identifying when the same entity occurs in negative scenarios

The results of this experiment are presented in Figure 4.6. In this experiment, we observe and measure results for the three platforms on pairs of different entities into which similarities have been injected, as presented in Figure 3.5 in Section 3.3.3. To ensure that the different entities have no common parts in the baseline scenarios, we have removed similar elements, such as primary keys. However, even though *1- F-measure* is 0 (Set *9* in Figure 4.6(a)), which means that none of the three platforms have matched the two different entities, incorrect matches between one of the two different entities and other entities may still be identified. The following can be observed:



(a) Expt 2: R1 (b) Expt 2: R2

Figure 4.6: Results of Experiment 2 for COMA++, Rondo and OpenII: Average (1-F-measure).

1. All three platforms have matched the two different entities (Figure 4.6(a)) when similarities have been injected into their names or their attributes (e.g., SNDE, SNSA, DNSA in Figure 3.5 in Section 3.3.3). COMA++ has performed the best among the three platforms, and only matched the two different entities in scenarios where they have the same names (Sets *13* to *16* in Figure 4.6(a)); OpenII has been least successful in this experiment as the two different entities have been matched in most scenarios. This is because all the three platforms always choose the top candidate matches for each element, and this also indicates that entity types are matched because they are more similar to each other than to other entity types but not because they represent the same real world notion.

2. COMA++ and Rondo perform satisfactorily in not matching different attributes (Figure 4.6(b)): where attributes are matched, this is normally because similar attributes have been introduced, and the remainder results from overlaps in the extents or the names of non-equivalent attributes. OpenII has matched several different attributes even in the baseline scenarios where no similarities have been injected. This shows that to select candidate attribute matches, the method of choosing a single match for an attribute that has the greatest similarity seems more reasonable than the method that does not restrict the number of matches associated with an attribute and keeps a candidate match as long as it is the top match for either of its associated attributes.

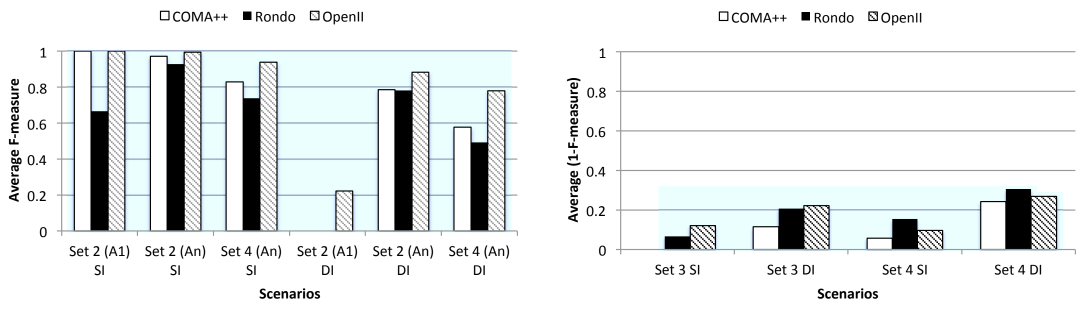
4.2.3 Experiment 3: Identifying where different names have been given to equivalent attributes in positive scenarios

The results of this experiment are presented in Figure 4.7(a). The graph distinguishes *SI* and *DI* cases, and includes cases where a single attribute (*A1*) or all attributes (*An*) have had their names changed to similar or different names. The following can be observed:

1. All three platforms have been generally successful at matching equivalent attributes in the SI case, regardless of the changes to their names and missing siblings. Rondo has occasionally matched an attribute to a different attribute where evidence between equivalent attributes is weak, e.g., when equivalent primary key attributes have different names and same instances (Set *2 (A1) SI* in Figure 4.7(a)).

2. In the DI case, better performance of the three platforms is reported where many attribute names have changed than in the case where a single attribute name is changed. This is because the ground truth involves only the attributes whose names have changed, and in the implementation of the experiment, where a single attribute name is changed (*A1*), this is often the primary key, where the disjoint instances at the tuple level give rise to completely disjoint instances at the attribute level, and thus there is no contribution to the matching of the attributes from the instance matcher.

3. OpenII is able to match equivalent attributes, even given a combination of disjoint instances and missing attributes (Set *4 (An) DI* in Figure 4.7(a)), whereas COMA++ and Rondo have performed slightly worse than OpenII. This because OpenII tries to keep every top match irrespective of its score, which corroborates the finding that OpenII emphasizes recall more than the other platforms.



(a) Expt 3 (b) Expt 4

Figure 4.7: Results of Experiments 3 and 4 for COMA++, Rondo and OpenII.

4.2.4 Experiment 4: Identifying where different names have been given to equivalent attributes in negative scenarios

The results of this experiment are presented in Figure 4.7(b). The following can be observed:

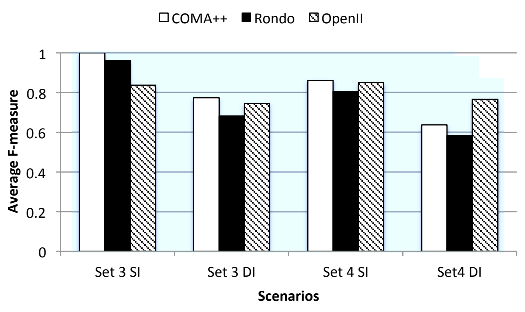
1. Some different attributes are matched that should not be, which can be explained as follows. In the collection of candidate matches identified by the three platforms, each element may be matched to several elements, supported by evidence, e.g., names and instances, therefore, an attribute that does not have a corresponding attribute is likely to be matched to some attribute due to coincidental name similarity or coincidental extent overlap. Thus, the idea of choosing the best candidate match for each element, though implemented differently by the three platforms, will always return some match for an element, resulting in incorrect matches between different attributes. COMA++ has performed slightly better than the other

two platforms because it applies the threshold of 0.1 and this helps to remove some incorrect matches.

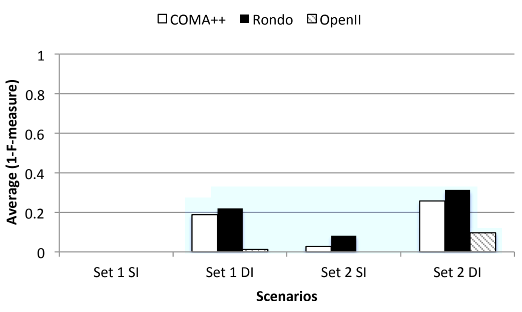
2. Some attributes that have a corresponding attribute are not associated with a match, thus being incorrectly identified as different attributes, in particular when match evidence between equivalent attributes is weak, e.g., due to disjoint instances, (all three platforms performed worse in Set *3 DI* than Set *3 SI* in Figure 4.7(b)). As there is little evidence supporting the match of equivalent attributes, all three platforms are likely to choose a different attribute as the best match.

4.2.5 Experiment 5: Identifying missing attributes in positive scenarios

The results of this experiment are presented in Figure 4.8(a). All three platforms are generally able to correctly identify missing attributes (i.e., identify them as attributes without correspondences) when equivalent entity types have the same instances (Sets *3 SI* and *4 SI* in Figure 4.8(a)), but have performed slightly worse in the disjoint instances case. Where missing attributes have been matched by the three platforms, this is due to the selection of the best candidate match for each element.



(a) Expt 5



(b) Expt 6

Figure 4.8: Results of Experiments 5 and 6 for COMA++, Rondo and OpenII.

4.2.6 Experiment 6: Identifying missing attributes in negative scenarios

The results of this experiment are presented in Figure 4.8(b). All three platforms have generally been successful at identifying that no attributes are missing, in particular when equivalent attributes have the same instances (Set *1 SI* and Set *2 SI* in Figure 4.8(b)). OpenII has performed significantly better than COMA++ and Rondo in that most non-missing attributes (i.e., attributes with correspondences) are matched to some attribute, because as long as an attribute is associated with a match (no matter whether it is correct or not) in the collection of candidate matches, OpenII will identify such an attribute as a non-missing attribute by associating it with a match in the final results.

4.2.7 Experiment 7: Identifying many-to-one attribute correspondences in positive scenarios

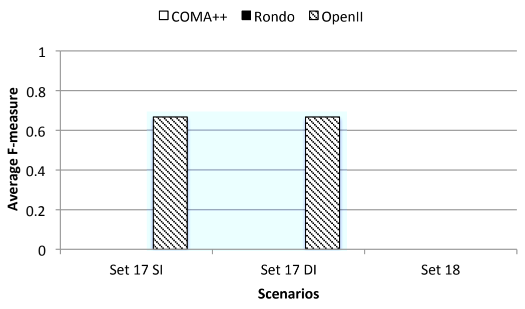
The results of this experiment are presented in Figure 4.9(a). The following can be observed:

1. COMA++ and Rondo have failed in this experiment. In contrast to Rondo, which only identifies one-to-one matches, the *Threshold+MaxDelta* method that COMA++ employs allows the identification of many-to-one matches. However, given the *delta* value of 0.01, the *MaxDelta* method sets a fairly small tolerance range below the top match of an attribute, thus only being able to return matches whose similarities are close to the top match. Nevertheless, a many-to-one attribute correspondence refers to a transformation of instances (e.g., string concatenation or numeric operation) between the *many* attributes and the *one* attribute rather than a selection of matches whose similarities are close, as determined by comparing names or instances of pairwise attributes. We anticipate that iMAP [DLD+04] could identify the many-to-one attribute correspondences. However, since the system is not publicly available, we have had no way to confirm or reject this assumption.

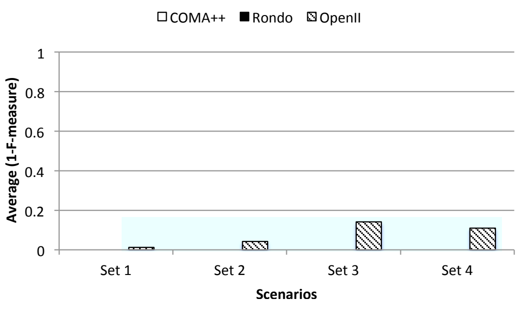
2. OpenII is the only platform that could identify many-to-one attribute correspondences for Types *2* and *3* presented in Section 3.3.4, where the *many* attributes and the *one* attribute have similar names (Sets *17 SI* and *17 DI*

in Figure 4.9(a)). This is because OpenII chooses a best match for each element but does not restrict that the element can only be associated with a single match, i.e., allowing a match to be kept as long as it is the best match for either of its associated elements. When the *many* attributes and the *one* attribute have similar names, the matches between the *many* attributes and the *one* attribute are usually the top matches for the *many* attributes, and thus are selected by OpenII.

3. Although OpenII has identified some many-to-one attribute matches, it does not demonstrate the ability to diagnose the many-to-one attribute conflicts, as the match evidence that OpenII uses comes from attribute names rather than instances. OpenII results have been significantly influenced by the changes of attribute names (the *F-measures* reported in Sets *17 SI* and *18 SI* in Figure 4.9(a) are completely different), but not by the changes of attribute instances (the *F-measures* reported in Sets *17 SI* and *17 DI* in Figure 4.9(a) are the same). OpenII cannot make use of the instance evidence because of the instance matcher plugged in: it only compares string similarity of extents of two attributes, but the instances of each of the *many* attributes and the *one* attribute are always different.



(a) Expt 7



(b) Expt 8

Figure 4.9: Results of Experiments 7 and 8 for COMA++, Rondo and OpenII.

4.2.8 Experiment 8: Identifying many-to-one attribute correspondences in negative scenarios

The results of this experiment are presented in Figure 4.9(b). The following can be observed:

1. COMA++ and Rondo do not identify many-to-one attribute correspondences where they should not, for the reasons stated in Section 4.2.7.

2. Due the best selection method implemented by OpenII, a few of incorrect many-to-one attribute correspondences have been returned.

4.2.9 Experiment 9: Identifying many-to-many entity correspondences in positive scenarios

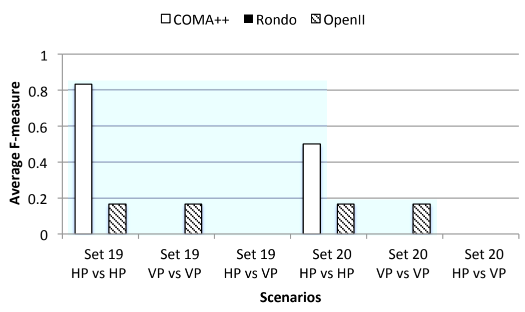
The results of this experiment are presented in Figure 4.10. Rondo is not able to achieve this task. COMA++ and OpenII have performed generally well in satisfying requirements *R2* and *R3*, which are, however, investigated only when the evaluated systems are able to meet requirement *R1*, as stated in Section 3.4.2, which neither of them meets. Therefore, none of the three platforms are able to diagnose many-to-many conflicts. Nevertheless, we still show the results of requirements *R2* and *R3* for the completeness of the evaluation. The following can be observed:

- *Requirement R1*, namely that each of the source entity types can be matched with all the alternatively fragmented entity types, can barely be satisfied by the three platforms (Figure 4.10(a) and (b)):

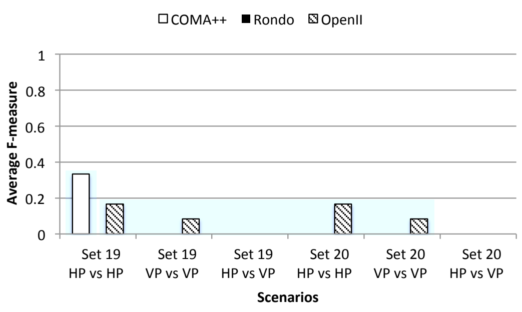
    1. COMA++ has only been able to associate the many-to-many entity types where the same instances are being represented in the horizontal partitioning models (Set *19 HP vs HP* and Set *20 HP vs HP* in Figure 4.10(a)), but has failed in other partitioning models or in disjoint instances. This is because only when the two original entity types that have the same instances are horizontally partitioned, are the similarities between each pair of entity types in the source and target sets close, and as such are selected by the *MaxDelta* method.

    2. Rondo is not able to identify any many-to-many entity correspondences.

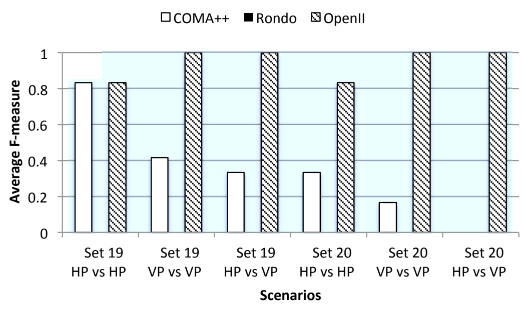
    3. OpenII has performed slightly better than the others. However, it has been fairly generous to associate a few of matches with an entity type (the *recalls* are always high, but the *precisions* are fairly low). Therefore, the patchy results shown by OpenII are because the equivalent



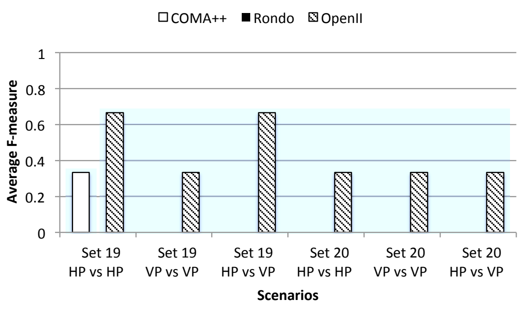
(a) R1, same instances



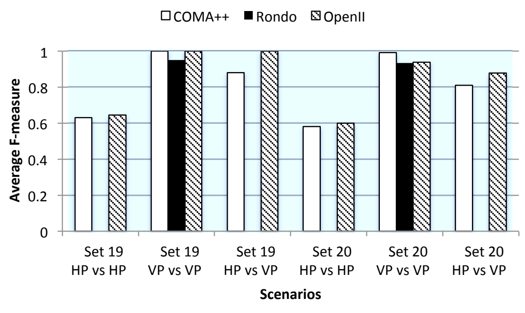
(b) R1, disjoint instances



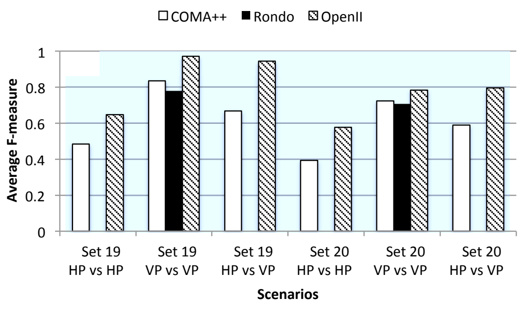
(c) R2, same instances



(d) R2, disjoint instances



(e) R3, same instances



(f) R3, disjoint instances

Figure 4.10: Results of Experiment 9 for COMA++, Rondo and OpenII: Average F-measure.

many-to-many entity types have also been matched to different entity types.

- *Requirement R2*, namely that primary key attributes should be matched, has been mostly satisfied by OpenII, and to a lesser extent by COMA++ (Figure 4.10(c) and (d)), whereas Rondo has failed in this task (we do not discuss it below):

    1. This requirement is partially satisfied by COMA++ when the alternatively fragmented entity types have the same instances and no changes have been made to attribute names, and thus the similarities of matches for many-to-many primary key attributes are close. However, there is a significant drop-off for disjoint instances (which are more disjoint at the attribute level for keys than for other attributes) or similar attribute names.
    2. OpenII has performed fairly satisfactorily in identifying the many-to-many primary key attributes in SI case, however, for cases where there is less evidence (e.g., the DI case) equivalent primary key attributes are matched to different attributes.

- *Requirement 3*, namely that appropriate correspondences can be identified between non-key attributes, has been satisfied quite well by COMA++ and OpenII, and has also been satisfied by Rondo where the original entity types have been vertically partitioned (Figure 4.10(e) and (f)):

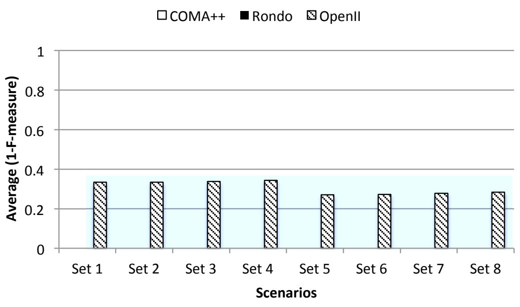
    1. COMA++ has been generally successful at matching non-key attributes in both scenario sets where the same instances are represented, but has performed slightly worse in the presence of disjoint instances. COMA++ has performed particularly well in the vertical partitioning scenarios (Set *19 VP vs VP* and Set *20 VP vs VP* in Figure 4.10(e)), as the non-key attributes only have single corresponding attributes; but has performed less well in the horizontal partitioning scenarios (Set *19 HP vs HP* and Set *20 HP vs HP* in Figure 4.10(e)) where many-to-many correspondences between non-key attributes should be identified. This indicates that COMA++ is more suited to identifying one-to-one correspondences than to many-to-many correspondences.

2. Rondo has performed well in matching non-key attributes with a single corresponding attribute (Set *19 VP vs VP* and Set *20 VP vs VP* in Figure 4.10(e) and (f)).

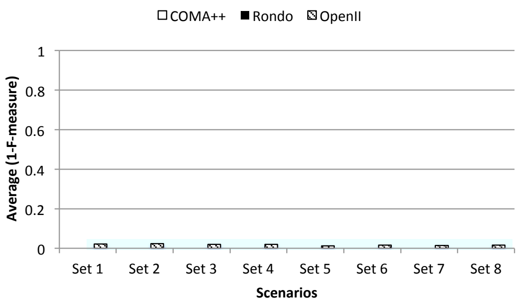
3. OpenII has been competitive with COMA++ in the SI case, but has performed better in the DI case (Figure 4.10(e) and (f)), as the lack of a threshold means that OpenII tends to return more matches, some of which are true positives.

4.2.10 Experiment 10: Identifying many-to-many entity correspondences in negative scenarios

The results of this experiment are presented in Figure 4.11. The following can be observed:



(a) Expt 10: R1



(b) Expt 10: R2

Figure 4.11: Results of Experiment 10 for COMA++, Rondo and OpenII: Average (1-F-measure).

1. COMA++ and Rondo have performed well in this experiment in that no many-to-many correspondences were identified for entity types in both source and target. However, again this also indicates that the two platforms have been designed for identifying one-to-one correspondences.

2. Although OpenII has not performed well in satisfying requirement *R1* in that several many-to-many entity correspondences have been identified (Figure 4.11(a)), it has been successful in satisfying requirement *R2* (Figure 4.11(b)), namely each attribute in each source entity type should not be matched to an attribute in each target entity type, and thus few many-to-many attributes have been identified. Therefore, we conclude that OpenII

has demonstrated reasonable ability to not identify many-to-many entity types in the negative scenarios.

4.3 Summary and Conclusions

In this chapter, we presented the results of an application of MatchBench to these state-of-the-art schema matching systems with a view to identifying whether they are successful in identifying correspondences between schemas in the presence of the schematic heterogeneities described by Kim *et al.* [KS91]. The work presented here differs from most reported evaluations of schema matching proposals, which mainly concentrate on the identification of lower-level one-to-one associations between individual schema elements. In contrast, their ability to combine these observations to draw higher-level conclusions has not been investigated. We summarize the lessons learnt from this application of MatchBench as follows:

- The schema matching systems used were designed to associate similar schema elements, and have been shown to perform rather better at this task than at diagnosing the schematic heterogeneities of Kim *et al.* [KS91], which provide a characterization of the relationships between the schemas that can be used to support mapping generation [MBPF09]. Schema elements tend to be matched by the existing approaches when they exhibited enough similarities, for example, when entity types have the same names or similar attributes, even though they may represent different real world notions.

- The schema matching systems used were designed for identifying one-to-one correspondences rather than many-to-many ones, as all three platforms have shown generally a good performance in Experiments *1*, *3*, *4*, *5* and *6* where only one-to-one correspondences were to be identified, but have failed in Experiments *7* and *9* where many-to-many correspondences were sought.

- The strategy of selecting candidate matches influences the overall performance of the schema matching system used. All three platforms employ the same basic idea that the best match of an element should always be chosen, though they have implemented the idea differently. The intuition behind this idea indicates that entity types (or attributes) are associated because they are more similar to each other than to other entity types (attributes) rather than because they are similar enough to be identified as representing the same real world notion, thus leading to the reporting of significant numbers of false positives in Experiment 2.

- COMA++ is a flexible schema matching platform, and offers alternative choices of matchers for different matching tasks. We anticipate that with more appropriate *threshold* and *delta* values, COMA++ would have performed better in experiments provided in MatchBench [SLDR05, LSDR07]. However, evaluating COMA++ with MatchBench brings out the well-known problem that setting any parameters generally requires access to at least some training data, and this presents practical challenges in certain applications. This observation in turn raises the question whether there is an opportunity to design a schema matching system that is less sensitive to choice of parameters. We note that, in the context of dataspaces, having to explore the parameter space of a matching system would be a significant upfront cost which dataspaces aim to drastically reduce.

- Designed as an interactive tool, OpenII seems unsuitable for scenarios where a huge number of matching tasks are required, such as MatchBench, where it is not practical for the user to manually choose matches for every single pair of schemas. It is also not suitable for scenarios that demand the automatic generation of matches, as even though we have used matches suggested by OpenII (i.e., the top matches), the *F-measure* reported in MatchBench experiments, e.g., Experiment *1*, is rather low. Furthermore, it is hard to obtain a threshold that helps to remove false positives from the past experience of OpenII developers.

- Instance-level data makes a strong contribution to the dependability of the matches, and there is little evidence that sophisticated composite matchers can compensate for shortcomings in the information provided by the leaves matcher, namely that when similarity of attributes is weak, equivalent entity types are hardly to be matched.

Lessons learnt from the application of MatchBench described in this chapter enable us to move forward with better awareness of the shortcomings of existing schema matching systems. The question now arise as to whether it is possible to devise a method for automatically inferring schematic correspondences without requiring interactive effort from users, whilst still obtaining results that are less sensitive to choice of parameters and improving shortcomings of the existing systems when evidence is inadequate.

Chapter 5

Inferring Schematic Correspondences

In this chapter, we propose an approach for inferring schematic correspondences between two schemas using an evolutionary search method, specifically a genetic algorithm. In contrast to most existing methods (e.g., [DR07, GYS07, MGMR02]) that only identify one-to-one correspondences between two schemas, the approach presented here is able to identify complex many-to-many correspondences that indicate the equivalence of sets of entities and of sets of attributes, in addition to one-to-one correspondences. We start with source and target schemas and the matches that denote their similarity, e.g., those produced by any of the matching approaches introduced in Chapter 2, and apply a genetic algorithm to search for a solution from a space of potential entity-level schematic correspondences between the two schemas. As required by the genetic algorithm, we design phenotype and genotype representations of a solution, and devise an objective function based on the vector space model [SWY75] in order to evaluate different solutions. Given a solution that represents a set of entity-level schematic correspondences and has the maximum fitness value assigned by the objective function of the genetic algorithm, we further identify attribute-level schematic correspondences.

The remainder of this chapter is structured as follows. We compare our approach with related work in Section 5.1. We present an overview of the approach in Section 5.2. To introduce the approach for inferring schematic correspondences at the entity-level, we present the inference framework, i.e., the genetic algorithm, in Section 5.3, and the representations of entity-level schematic correspondences in Section 5.4, followed by an objective function that calculates the fitness value of the entity-level schematic correspondences in Section 5.5. We illustrate the method for identifying attribute-level schematic correspondences in Section 5.6. We conclude the chapter in Section 5.7.

5.1 Related Work

5.1.1 Semantic relationships between data sources

The relationships specified by schematic correspondences aim to capture more semantics than the matches identified by most of the existing schema matching approaches. Table 5.1 characterizes correspondences that associate schema elements as *simple* correspondences and *complex* correspondences, and divides each of them into four types: i) *1-to-1 entity-level* correspondences, ii) *1-to-1 attribute-level* correspondences, iii) *n-to-m entity-level* correspondences, and iv) *n-to-m attribute-level* correspondences, thus resulting in the following eight types:

	simple correspondences	*complex* correspondences
1-to-1 entity-level	$Type_1^s$	$Type_1^c$
1-to-1 attribute-level	$Type_2^s$	$Type_2^c$
n-to-m entity-level	$Type_3^s$	$Type_3^c$
n-to-m attribute-level	$Type_4^s$	$Type_4^c$

Table 5.1: Comparison of semantic relationships.

- $Type_1^s$ and $Type_2^s$, refer to the matches that associate 1-to-1 entities or 1-to-1 attributes, and identify the relationship of *equivalence* (e.g., [DR07, MBR01, MGMR02, KN08, MBDH05, DDH01, BN05].

- $Type_3^s$, denotes the *equivalence* relationship between two sets of entities, as can be identified by Xu *et al* [XE06].

- $Type_4^s$, indicates that two sets of attributes are equivalent, and can be generated by several approaches, such as iMAP [DLD+04], Dai *et al.* [DKS+08], Xu *et al.* [XE06] and Warren *et al.* [WT06].

- $Type_1^c$ and $Type_2^c$, offer more semantics than $Type_1^s$ and $Type_2^s$ between 1-to-1 entities and 1-to-1 attributes, such as *Is-a* and *Has-a* relationships which can be identified by SeMap [WP08], *subsumption* and *intersection* relationships as presented by N. Rizopoulos [Riz04], and *more general* and *less general* relationships as presented by Giunchiglia *et al.* [GYS07].

- $Type_3^c$, not only denotes the *equivalence* relationship between two sets of entities, but also specifies the operators (e.g., *join*) that connect the entities within each set (e.g., the indirect matches generated by Xu *et al.* [XE06]).
- $Type_4^c$, refers to the *equivalence* relationship at the attribute-level that also indicates the association of attributes within a set, e.g., *concatenation.* Approaches that identify $Type_4^s$ have also been able to generate $Type_4^c$ [DLD+04, DKS+08, XE06, WT06].

The research conducted in this thesis contributes to the identification of *complex* correspondences, including $Type_1^c$ (e.g., *Different Names for Same Entities*), $Type_2^c$ (e.g., *Different Names for Same attributes*), $Type_3^c$ (i.e., *many-to-many entity correspondences*) and $Type_4^c$ (i.e.., *many-to-many attribute correspondences*), as stated in Section 1.1.2.

5.1.2 Methods for inferring complex correspondences

SeMap [WP08] is a system that identifies rich semantic relationships (e.g., *Has-a*, *Is-a*, *Associates* and *Equivalent*) between elements of different data models (e.g., relational, XML, RDF, HTML), which map to $Type_1^c$ and $Type_2^c$ in Table 5.1. It starts from a collection of initial matches that represent the equivalence relationships between elements of two schemas, and are produced by ordinary matchers introduced by Chapter 2, such as *name*, *type*, *structure*, or *instances* matchers. Selecting from the initial matches, SeMap then assigns a match to each element in both source and target schemas, thus obtaining two sets of candidate matches. It selects an optimal set from the two sets of candidate matches, based on which it then applies heuristic rules to infer the rich semantic relationships between the elements of two schemas as the final results. For example, SeMap infers *Is-a* or *Has-a* relationships of two elements by considering prefix/suffix of element names (e.g., grad-TA is a TA) and by considering instance subsumption. Both SeMap and the research presented in this thesis infer complex relationships between schema elements from matches, but differ in the following ways: i) they differ on the specific relationship types (SeMap infers *Has-a*, *Is-a*, *Associates* and *Equivalent* relationships, and our method infers the *Equivalent* relationship), and thus SeMap can be seen as complementary to our method; ii) SeMap only identifies 1-to-1 relationships, whereas our approach can infer both 1-to-1 and n-to-m relationships, in which identifying the n-to-m relationships has been considered as a difficult problem; iii) SeMap requires various parameters to be set during the identification, such as weights for combining the results of the matchers or thresholds for selecting candidate matches (however, the authors never mention the threshold values, which are actually decisive to the selection of matches, as we have learnt from using MatchBench to evaluate various matching frameworks in Chapter 4). By contrast, our method applies a search method to infer schematic correspondences, and thus does not require context-specific parameters, as will be discussed in Sections 5.3 to 5.5.

Xu *et al.* [XE06] propose a semi-automatic approach that infers 1-to-1 and n-to-m complex relationships between schema elements. Specifically, the approach is able to infer all types of complex relationships, i.e., $Type_1^c$ to $Type_4^c$ in Table 5.1. In addition to identifying that two sets of elements are equivalent, Xu *et al.* also apply operators over elements in the source (or target) set that specify their associations further, thus expressing the n-to-m relationships in the form of an algebra. The operators supported by the method include, the following standard operators inspired by the relational algebra: *Selection*, *Union* , *Join*, *Projection*, and in addition some specific operators, such as *Composition* and *Decomposition*. As we have introduced the specific techniques employed by Xu *et al.* in Section 2.4, we only compare them with our method here: i) Xu *et al.* make significant use of domain specific ontologies, which are not always available in real world scenarios, and sometimes require an expert to provide specific information, while our method does not rely on such external resources or human effort; ii) Xu *et al.* utilize specific constraints of the employed schema model (i.e., the conceptual model [Emb97, LEW00]) to infer the element associations in the source (or target) set, whereas we do not make use of any constraints of the relational model (e.g., primary and foreign key information) to identify the horizontal or vertical partitioning, and thus our method can be easily extended to wider contexts where schema constraints are not available; iii) a single element may be a member of several complex relationships in the approach presented by Xu *et al.*, thus requiring the user to choose the desired relationships, whilst our method only associates an element with a single correspondence.

Rizopoulos [Riz04] and **Giunchiglia *et al.*** [GYS07] present approaches for identifying complex 1-to-1 relationships between elements (i.e., $Type_1^c$ and $Type_2^c$ in Table 5.1), such as *equivalence*, *subsumption*, *intersection*, *incompatibility*, *disjointness* [Riz04], and *equivalence*, *more general*, *less general*, *disjointness*

[GYS07]. Rizopoulos compares the instance containment of elements to derive the relationships, while Giunchiglia *et al.* infer such relationships by determining the element name containment using WordNet [Mil95], e.g., *synonym* or *hypernym/hyponym* (*is-a* relationship). The work by Rizopoulos and Giunchiglia *et al.* are complementary to our approach, and could be used to improve the quality of the matches used as input by our method.

iMAP [DLD$^+$04], **Dai *et al.*** [DKS$^+$08] and **Warren *et al.*** [WT06] specialize in discovering complex n-to-m relationships at the attribute-level using instance data (i.e., $Type_4^s$ and $Type_4^c$ in Table 5.1). iMAP [DLD$^+$04], as presented in Section 2.4, detects various complex attribute matches using different preset formulae that transform instances between the source and target attributes. Dai *et al.* [DKS$^+$08] contribute to the identification of n-to-1 attribute matches where concatenation of the *n* attributes is equivalent to the single attribute. Their method can handle the case that the *n* attributes have disjoint instances with the single attribute. The approach proposed by Warren *et al.* [WT06] also identifies n-to-1 matches for string attributes, and meanwhile creates a transformation formula that concatenates the *n* attributes whose cardinality is unknown in advance into the single attribute. Our method contributes more to the inference of complex relationships at the entity-level, though we implement a simple method for inferring n-to-m attribute relationships. Thus, iMAP [DLD$^+$04], and methods proposed by Dai *et al.* [DKS$^+$08] and Warren *et al.* [WT06] can be seen as complementary to our method.

Melnik *et al.* [MBHR05] implement a simple method for automatically generating views between tables of two relational schemas for Model Management 2.0 [BM07], as introduced in Section 2.5.2. This method takes as input two relational schemas and matches, and associates two sets of tables using relational algebras (e.g., $\pi_{att_1,att_2}(table_1 \bowtie table_2) = \pi_{att'_1,att'_2}(table'_1 \bowtie table'_2)$), which map to $Type_1^c$ in Table 5.1. However, this method makes the strong assumption that each table has a primary key and that each foreign key-primary key join is lossless, which may not always hold in real world schemas; our method does not make use of key information, and thus can be easily extended to real applications where it could be missing.

5.1.3 Identifying correspondences as a search problem

Our method is not the first one that casts the schema matching problem as an evolutionary search problem. The method proposed by **Elmeleegy *et al.*** [EOE08] also approaches the problem from this angle, but only contributes to the identification of 1-to-1 attribute-level relationships. The method uses information extracted from query logs, such as the occurrence and position of an attribute in query clauses (e.g., `select-from-where`), to identify 1-to-1 equivalent attributes. Their method adapts the scoring functions proposed by Madhavan *et al.* [MBDH05] to calculate the similarity of attributes from the query logs, and employs a genetic algorithm to select a set of 1-to-1 attribute matches that have the highest similarity. We apply the same approach as Elmeleegy *et al.*, i.e., a genetic algorithm, to infer schematic correspondences, but have designed novel scoring functions to achieve our target: inferring equivalent 1-to-1 and n-to-m entities and discovering the particular kinds of associations of entities, i.e., horizontal or vertical partitioning, which, to the best of our knowledge, has not been done before. In addition, it has been recognized that suitable usage data, such as the query logs employed by Elmeleegy *et al.*, is hard to obtain [BBR11]. By contrast, our method is able to make use of information that tends to be more readily available, such as schemas, instances and data types.

5.1.4 Identifying matches without applying parameters

Hong *et al.* [HHB10] propose a method for identifying 1-to-1 attribute matches between source and target query interfaces supported by Web databases. Given a particular application domain (e.g., flight booking), the two query interfaces usually represent similar information but are represented differently. The approach proposed by Hong *et al.* implements three individual matchers to identify semantic similarity, namely edit distance and Jaro distance of attribute names, and a matcher to identify the data type similarity of attributes. The approach is able to combine evidence produced by different matchers using Dempster-Shafer theory (for reasoning with uncertain, imprecise and incomplete information [Sha76]), which does not require any weight parameters. Based on the combined similarities of all possible pairwise attributes, the approach selects the top k candidate matches for each source attribute, which are further chosen into the result 1-to-1 attribute matches between the two query interfaces. Our method is similar to the one proposed by Hong *et al.* in that both methods consider matches produced by various matchers as sources of evidence and try to identify correspondences without using parameters. However, they are different in that: i) our method focuses on inferring schematic correspondences at both entity and attribute-levels, and thus deals with a more difficult problem than the approach proposed by Hong *et al.* that only identifies 1-to-1 attribute matches; and ii) as we are facing a more complicated challenge, we chose to utilize matches produced by the existing schema matching approaches rather than implementing these matchers again, and we also used a simpler method to combine various sources of evidence than the approach proposed by Hong *et al.*, whereas the latter has contributed significantly in terms of the technique for combining evidence, which can be used to improve the input of our method.

5.2 Overview of the Approach

As motivated in Chapter 1, the aim of the work presented in this thesis is to develop a method for automatically inferring schematic correspondences between two schemas in order to bootstrap dataspace management systems. To achieve this aim, the following four requirements have to be fulfilled by the designed method:

1. as we aim to infer the correspondences as automatically as possible and require as little user involvement as possible, it should not return a collection of candidate schematic correspondences for an element and ask the user to choose a desired one, and as such schematic correspondences should be disjoint, i.e., an element participates in at most a single schematic correspondence;

2. it should not require the user to set context-specific parameter values (e.g., thresholds to select correspondences) before executing the function, because setting such parameters usually needs training data, which is hard to obtain by the user;

3. it should not assume that external resources (e.g., domain specific ontologies) are available; and

4. it should not assume that schemas contain referential constraints, as they may be missing in real world applications, e.g., web tables [CHW$^+$08].

Significant work has been done on schema matching approaches (e.g., [DR07, MGMR02, MBDH05, DMDH02, DDH01, EM07]), and thus we decided to use their representation of the similarity of schema elements (e.g., name or instance similarity) along with the two schemas as input to the approach we present here. Furthermore, we do not require any additional information, such as external resources or referential constraints, for the inference of schematic correspondences.

Specifically, our method takes as input a pair of schemas and matches between them that are identified by existing schema matching systems (e.g., COMA++ [DR07]), and produces schematic correspondences between the input schemas. To illustrate our method, let us reuse the relational databases RDB1 and RDB2 in Section 1.1.2 as a running example, where primary and foreign keys are removed to fit in with our assumption, as presented in Example 1.

Example 1 ***Relational databases.***
RDB1:
home_cust (id, name, birth, a_id, p_city, p_area, p_local)
overseas_cust (id, name, birth, a_id, p_city, p_area, p_local)
account (id, name, balance, tax)
RDB2:
customer (key, c_fname, c_lname, c_birth, account_key)
cust_phone (key, city, area, local, extension)
cust_account (key, account_name, account_balance)

Figure 5.1 is an example that visually describes the input and output of our method, where Figure 5.1(a) presents an example for the input matches between RDB1 and RDB2, and Figure 5.1(b) describes the resulting schematic correspondences.

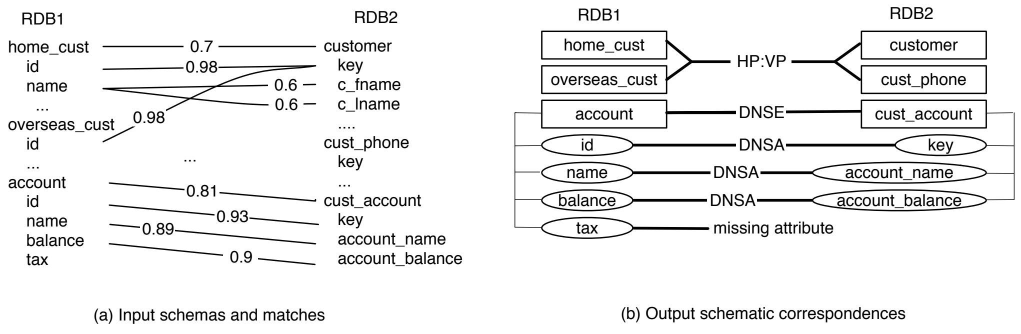


Figure 5.1: Examples for input and output of our method.

Our method for inferring schematic correspondences consists of three steps, as described in Figure 5.2. The method takes as input a pair of schemas, e.g., RDB1 and RDB2 above, and matches identified by existing schema matching systems, e.g., COMA++ [DR07]. It then goes through three steps, as presented in detail in Algorithm 1, and produces schematic correspondences between the two input schemas as the output.

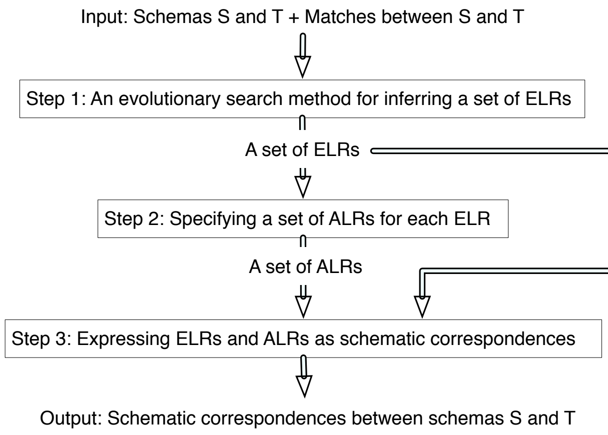

Input: Schemas S and T + Matches between S and T
Step 1: An evolutionary search method for inferring a set of ELRs
A set of ELRs
Step 2: Specifying a set of ALRs for each ELR
A set of ALRs
Step 3: Expressing ELRs and ALRs as schematic correspondences
Output: Schematic correspondences between schemas S and T

Figure 5.2: Overview of the approach.

In the first step (Line *1* in Algorithm 1), we obtain a set of *disjoint* entity-level relationships (*ELRs*) between RDB1 and RDB2, e.g., ELR_1 and ELR_2 in Example 2. An *ELR* associates a pair of entity sets. ELR_2 represents a pair of 1-to-1 entities and indicates that they are equivalent, thus representing the same real world concept. ELR_1 associates n-to-m entities, which not only denotes the equivalence relationship between its associated sets of entities but also carries information about the nature of the entity association in each set. In particular, home_cust and overseas_cust are horizontally partitioned (*HP*), and customer and cust_phone are vertically partitioned (*VP*). Disjointness of the two *ELRs* means that each source or target entity only participates in a single schematic correspondence as mentioned previously[1].

Example 2 ***Disjoint entity-level relationships between*** **RDB1** ***and*** **RDB2*.***
ELR_1 = ({home_cust, overseas_cust}, {customer, cust_phone}), *HP vs VP*
ELR_2 = ({account}, {cust_account})

[1]For the remainder of this thesis, whenever we mention a set of *ELRs*, we usually refer to a set of *disjoint ELRs* between source and target schemas, unless stated otherwise.

Algorithm 1 Inference(Schema $\mathcal{S}$, Schema $\mathcal{T}$, sets of matches $\{M_1, ..., M_\eta\}$)

```
1: identify a set of disjoint entity-level relationships $\{ELR_1, ELR_2, ..., ELR_n\}$;
2: for each $ELR_i \in \{ELR_1, ELR_2, ..., ELR_n\}$ do
3:     identify a set of disjoint attribute-level relationships $\{ALR_1, ALR_2, ..., ALR_m\}$;
4: express $\{ELR_1, ELR_2, ..., ELR_n\}$ and $\{ALR_1, ALR_2, ..., ALR_m, ...\}$ as schematic
   correspondences.
```

In the second step (Lines *2* and *3* in Algorithm 1), we identify a set of disjoint attribute-level relationships (*ALRs*) for each *ELR* identified in the first step. For ELR_2 in Example 2, we identify a set of disjoint *ALRs* as presented in Example 3. Similar to an *ELR*, each *ALR* represents an equivalence relationship of its associated attribute sets.

Example 3 ***Disjoint attribute-level relationships for*** ELR_2**.**
ALR_1 = ({account.id}, {cust_account.key}),
ALR_2 = ({account.name}, {cust_account.account_name})
ALR_3 = ({account.balance}, {cust_account.account_balance})

In the last step (Line *4* in Algorithm 1), we express the identified *ELRs* and *ALRs* as schematic correspondences as described in Section 1.1.2. Schematic correspondences transformed from ELR_1 and ELR_2 (Example 2) and from ALR_1 to ALR_3 (Example 3) are presented in Example 4.

Example 4 ***Example schematic correspondences between*** RDB1 ***and*** RDB2**.**
{home_cust, overseas_cust} *and* {customer, cust_phone} *are HP vs VP;*
account *and* cust_account *are Different Names for the Same Entities (DNSE);*
account.tax *is a missing attribute of* cust_account*;*
account.id *and* cust_account.key *are Different Names for the Same Attributes (DNSA);*
account.name *and* cust_account.account_name *are DNSA;*
account.balance *and* cust_account.account_balance *are DNSA.*

We cast the first step in Algorithm 1, i.e., the problem of inferring *ELRs*, as a *search* and *optimization* problem, where the *search* refers to the process of exploring the space of potential sets of *ELRs* and the *optimization* indicates that search seeks to maximize on the value of an objective function. In our context, we aim to identify a set of *ELRs* that associate 1-to-1 or n-to-m entities (a *search* problem), and such that each *ELR* that represents an equivalence relationship has as high a similarity score as possible (an *optimization* problem). We call such a set of *ELRs* a **solution**. Casting the problem as a search and optimization problem allows different solutions to compete with each other, and thus there is no need to apply context-specific heuristic rules (Requirement 2 above), e.g., thresholds.

As the search space of all possible solutions between source and target schemas is very large, an exhaustive search may be very time consuming and expensive, and, therefore, infeasible. Other search approaches, e.g., hill climbing, local search and simulated annealing, explore the search space based on a single solution, and the quality of the initial solution influences the quality of the returned solution significantly. In the context of inferring *ELRs*, no preferred solution can be used as the initial solution, and thus searching for an optimal solution using those approaches may be time consuming. The evolutionary algorithm is a population-based search method that relies on a collection of solutions rather than a single solution for future exploration, thus using much less time to complete the task than other search methods [MF04]. It also visits the space more efficiently, by not only exploring neighborhoods of a single solution (using a *mutation* operator) but also by examining the neighborhoods of pairwise solutions (using a *crossover* operator), and thus the quality of the initial solutions is not decisive in terms of finding the best result. Furthermore, this algorithm adopts the survival-of-the-fittest insight from evolution theory and only passes the best solutions down to the next generation, which in turn are used to generate new solutions, thus helping to reach an optimal solution faster than other search methods. For these reasons, we decided to employ an evolutionary algorithm, specifically a genetic algorithm [ES03], to search for a particular set of *ELRs*.

In the following sections, we present the techniques required for the first step (the identification of a set of *ELRs*), including the search framework employed by the genetic algorithm, the representations of *ELRs* and the objective function that models our requirement for identifying *ELRs*, in Sections 5.3 to 5.5, respectively. We describe the method used in the second step for identifying *ALRs* in Section 5.6. As transforming *ELRs* and *ALRs* into schematic correspondences is a fairly easy step (also see Example 4), we do not discuss it here.

5.3 A Framework for Searching Entity-Level Relationships

Before moving on to describe the search framework used by the genetic algorithm, we first introduce some terms defined by Eiben *et al.* [ES03] that are necessary to understand the framework. In the context of this thesis, a **solution** represents a set of disjoint *ELRs* between source and target schemas. Usually, a solution has *phenotype* and *genotype* representations that represent the solution in the real world context and in the problem-solving space, respectively. In particular, we design the phenotype representation and adopt a binary string as the genotype representation for a set of *ELRs*, as illustrated in Section 5.4. A **generation** forms the basic unit of the genetic algorithm and holds a set of solutions, known as a **population**. The **population size** represents the number of solutions in a population (or generation), and is fixed to a chosen size during the whole search process. We set the population size as 30, as empirical studies have shown that, for binary genotype representations, population sizes as small as 30 are quite adequate in many cases [SCED89, Gre86]. A **parent** is a solution in a population that has been chosen to go through mutation and crossover operators to reproduce new solutions. **Offspring** are the newly generated solutions resulting from mutation and crossover. The **objective function** usually models the requirements of the proposed search problem and assigns a fitness value to a solution to indicate its relative quality among all solutions with respect to the requirements.

Population size, as well as other parameters mentioned in this section, are a part of the evolutionary search. In principle, we follow the literature on the genetic algorithm to configure the parameters, whose settings do not change in different contexts, and therefore, are not context-specific.

Algorithm 2 Genetic Algorithm

```
1: initialize a population with random solutions;
2: evaluate each solution in the population by the objective function;
3: repeat
4:     select parents;
5:     reproduce offspring by applying mutation and crossover operators on the parents;
6:     evaluate offspring by the objective function;
7:     select survivors as the next generation;
8: until termination condition is satisfied
```

As described in Algorithm 2 [ES03], a genetic algorithm usually begins with an initial population composed of a set of randomly generated solutions, each of which is assigned a fitness value using the objective function (defined in Section 5.5). This is followed by an iteration of the search process (as stated in Lines *4* to *7* in Algorithm 2) until a termination condition is satisfied. In the case of our problem of inferring schematic correspondences from matches, there is not a known optimal fitness level to terminate the algorithm, i.e., a fitness value that we anticipate an optimal solution to have or exceed. In such a case, Eiben *et al.* [ES03] suggest applying an upper limit of, such as some elapsed CPU time or some total number of fitness evaluations, to the search process to guarantee it eventually stops at some point. We choose the termination condition as the parameter **iterations**, which specifies the number of times the search process should be repeated, as it is the simplest parameter to control. In particular, the following steps are completed at each search iteration. We usually manipulate solutions in their phenotype representations to each of the steps, unless stated otherwise.

- **Selecting parents** selects from the current generation those that should be used as parents for the next generation with the aim of improving the quality of subsequent generations over time [ES03]. Roulette-wheel selection is applied in this step, as using this technique solutions whose fitness values are comparatively high in the generation have a higher chance of being chosen, and thus serve as the parents solutions to reproduce offspring via mutation and crossover operators. It maps the fitness value f_i of each solution i in the current generation (μ solutions in total) to a probability value $P_{sel}(i) = f_i / \sum_{j=1}^{\mu} f_j$. For the 1 to μ solutions, it then calculates a list of values $[a_1, ..., a_\mu]$ where $a_i = \sum_{j=1}^{i} P_{sel}(j)$. Roulette-wheel selection picks a random float value r from [0, 1], and sequentially scans solutions 1 to μ until it finds a solution j whose corresponding value $a_j > r$.

- **Mutation** is a unary variation operator that manipulates a single parent **genotype** (a binary string) and is used to improve the diversity of a population [BR03]. It randomly generates a number in the range of [0, 1] for each bit in the genotype, and compares it with the value of a parameter *mutation rate* p_m in order to decide whether this particular bit should be changed (inverted in our case from 0 to 1 or from 1 to 0). If the random number generated for a bit is smaller than p_m, its corresponding binary value in the genotype is inverted [ES03]. We choose to set $p_m = 1/n$ (n is the length of the genotype), because it produces good results for a wide variety of test functions as suggested in [MSV93].

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|c|c|c|c|}\hline 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array}$$

- **Crossover** is a binary variation operator that merges information from two parent **genotypes** into two offspring genotypes [ES03], and explicitly tries to combine "good" parts of the two parent solutions [BR03]. We apply the simplest one-point crossover, namely one that chooses a random number x in the range of $[0, n-1]$ (n is the length of the genotype), splits the two parents at the x^{th} bit, and exchanges the tails of the two parents starting from the x^{th} bit.

$$\begin{array}{c} \begin{array}{|c|c|c|c|c|c|c|c|}\hline 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array} \\ \\ \begin{array}{|c|c|c|c|c|c|c|c|}\hline 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline \end{array} \end{array} \xrightarrow{x=2} \begin{array}{c} \begin{array}{|c|c|c|c|c|c|c|c|}\hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline \end{array} \\ \\ \begin{array}{|c|c|c|c|c|c|c|c|}\hline 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array} \end{array}$$

  Crossover is usually applied probabilistically according to the *crossover rate* p_c. It chooses a random number between 0 and 1, and if the number is smaller than p_c, the operator is applied on the parents to reproduce the offspring; otherwise the parents are copied into the offspring. Given a small population (with 20 - 40 solutions), good performance is associated with a high *crossover rate* $p_c \approx 0.9$ [SCED89, Gre86].

- **Selecting survivors** determines the next generation by choosing μ survivors (μ is the population size and is set to 30) that have the highest fitness values from μ parent solutions in the last generation and λ offspring that is produced from the μ parent solutions using mutation and crossover operators, where μ represents the population size. It is sometimes suggested that λ should be set to μ [ES03].

5.4 Representations of Entity-Level Relationships

Before introducing the representations of a solution (i.e., a set of *ELRs*), we define schemas (Definition 1) and matches (Definition 2) as follows.

Definition 1 ***Schema.*** **A schema** *$\mathcal{S}$ is composed of a set of entities (e.g., tables) $\{S_1, S_2, ..., S_\mu\}$, where each S_i ($i = 1, ..., \mu$) contains a set of attributes $\{S_i.A_1, S_i.A_2, ..., S_i.A_\alpha\}$. A construct is either an entity or an attribute in a schema $\mathcal{S}$.*

Definition 2 ***Match.*** **A basic match** *m between schemas $\mathcal{S}$ and $\mathcal{T}$ is a triple $\langle s, t, \delta\rangle$ where s is a construct in $\mathcal{S}$, t is a construct in $\mathcal{T}$ (note that, a match only associates either two entities or two attributes here), and δ is a float value in the range of $[0, 1]$ that indicates the similarity of s and t (0: completely different; 1: exactly the same). A set of basic matches $M_k = \{m_1^k, m_2^k, ..., m_\gamma^k\}$ associates each two constructs from $\mathcal{S}$ and $\mathcal{T}$, e.g., $m_i^k = \langle s_i^k, t_i^k, \delta_i^k\rangle$ ($i = 1, ..., \gamma$), which are identified by the k^{th} matcher. In particular, $\gamma = e_s \times e_t + a_s \times a_t$, where e_s and e_t represent the numbers of entities in $\mathcal{S}$ and $\mathcal{T}$, respectively, and a_s and a_t represent the total numbers of attributes in $\mathcal{S}$ and $\mathcal{T}$, respectively. Assuming there are η matchers, we have η sets of matches $M_1, M_2, ..., M_\eta$, from which we derive* **a new set of matches** *$\mathcal{M} = \{m_1, m_2, ..., m_\gamma\}$, where $m_i = \langle s_i, t_i, \Delta_i\rangle$ ($i = 1, ..., \gamma$) is* **a derived match***, in which $\Delta_i = \sum_{k=1}^{\eta} \delta_i^k$ is a float value in the range of $[0, \eta]$.*

As we are dealing with a more complicated problem, i.e., inferring schematic correspondences, we decided to pay as little attention to the input matches as possible. We do not wish to use weights to combine basic matches produced by different matchers, and thus we simply choose to sum up their similarity scores, as described in Definition 2. However, we anticipate that an effective method for combining evidence without using weights, such as the method proposed by Hong *et al.* [HHB10], would be helpful to improve the quality of match evidence used in this chapter (i.e., Definition 2). In the following discussion, we use the term **match** to refer to **the derived match**. And the derived matches are used as the input of the evolutionary search algorithm, together with schemas $\mathcal{S}$ and $\mathcal{T}$.

In an evolutionary algorithm (EA), the same solution may be represented differently by its *phenotype* and *genotype*, which map to the solution in the real world context and in the problem-solving space (i.e., EA space), respectively [MF04]. Usually, the phenotype representation of a solution is problem-specific and is necessary for designing an EA, while the genotype representation is the encoding of the phenotype representation within the EA and is optional for the design. Options for the genotype representation are binary representations, integer representations, permutation representations, etc. [ES03]. While designing the EA, we decided to also include the genotype for a solution in addition to the phenotype, because we can then adopt the canonical mutation and crossover operators for the genotypes during the search (Section 5.3) rather than designing the operators for the phenotype representations by ourselves, thereby reducing the complexity of the algorithm and increasing the trust it can command. Specifically, we employ the simplest genotype representation (i.e., the binary string), as it is recommended to select the smallest alphabets to represent a chromosome that permits a natural expression of the problem [Gol89]. We define the *phenotype* and *genotype* representations of a solution in Definitions 3 and 4, respectively.

Definition 3 ***Phenotype.*** *Given schemas $\mathcal{S}$ and $\mathcal{T}$, and matches $\mathcal{M}$, the* **phenotype** $\mathcal{P}$ *of a solution is defined as a set of ELRs* $\{ELR_1, ..., ELR_n\}$, *where* $ELR_i = (ES_i^s, ES_i^t)$, $i = 1, ..., n$. ES_i^s *and* ES_i^t *are source and target entity sets with cardinalities*$\geqslant$*1, respectively, and satisfy the following conditions: i) there exists at least either an entity-level match or an attribute-level match* $m \in \mathcal{M}$ *between each entity* $S_a \in ES_i^s$ *and each entity* $S_b \in ES_i^t$*; and ii) for each* $ELR_{i'} = (ES_{i'}^s, ES_{i'}^t) \in \mathcal{P}$, $i' \neq i$, $ES_i^s \cap ES_{i'}^s = \emptyset$ *and* $ES_i^t \cap ES_{i'}^t = \emptyset$. *Specifically, entity sets* ES_i^s *and* ES_i^t *are* **associated by** $ELR_i \in \mathcal{P}$, $i = 1, ..., n$. *Each entity* $S_j \in \mathcal{S}$ *(or* $\in \mathcal{T}$*) that satisfies* $S_j \notin \cup_{i=1}^n ES_i^s$ *(or* $\notin \cup_{i=1}^n ES_i^t$*) is called an* **unassociated entity** *of* $\mathcal{P}$.

Considering Example 1 again, we present the derived matches between RDB1 and RDB2 in Figure 5.3, where a set of matches, irrespective of whether they are at the entity-level or the attribute-level, between two entities is represented by a line. Examples for phenotypes are presented in Example 5. Given $\mathcal{P}_2$, {overseas_cust} and {cust_account} are *associated by* $\mathcal{P}_2$; the remaining entities, e.g., home_cust and customer, are *unassociated entities* of $\mathcal{P}_2$.

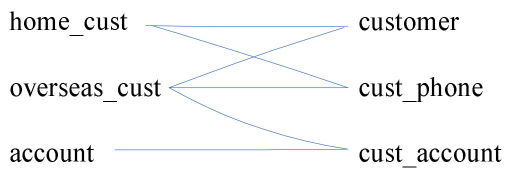


Figure 5.3: Matches between RDB1 and RDB2.

Example 5 Phenotypes.
$\mathcal{P}_1$={({home_cust, overseas_cust}, {customer, cust_phone}), ({account}, {cust_account})}
$\mathcal{P}_2$={({overseas_cust}, {cust_account})}

$\mathcal{P}_3$={({home_cust}, {cust_phone}), ({overseas_cust}, {customer})}
$\mathcal{P}_4$={({home_cust, overseas_cust}, {customer, cust_phone})}
$\mathcal{P}_5$={({home_cust}, {customer}), ({overseas_cust}, {cust_phone}),
({account}, {cust_account})}

Definition 4 ***Genotype.*** *Given schemas $\mathcal{S}$ and $\mathcal{T}$, and matches $\mathcal{M}$, a sequence of* **chromosomes** *is defined as $\mathcal{C} = \langle c_1, c_2, ..., c_m \rangle$, where $c_i = (S_a, T_b)$ is a pair of source and target entities ($i = 1, ..., m$), between which there exists at least either an entity-level match or an attribute-level match $m \in \mathcal{M}$. The* **genotype** *of a solution is defined as a sequence of binary values $\mathcal{G} = \langle x_1, x_2, ..., x_m \rangle$, where $x_i \in \{0, 1\}$ ($i = 1, ..., m$). In particular, $x_i = 1$ (or 0) represents that the i^{th} pair of entities in the sequence of chromosomes $\mathcal{C}$, i.e., $c_i = (S_a, T_b)$, is (or is not) associated in the solution.*

Following on with our example, we show the sequence of chromosomes in Example 6, which is fixed throughout the whole search process, and genotype representations for phenotypes $\mathcal{P}_1$ to $\mathcal{P}_5$ in Example 7.

Example 6 Chromosome.
$\mathcal{C}$ = ⟨(home_cust, customer), (overseas_cust, customer), (home_cust, cust_phone), (overseas_cust, cust_phone), (overseas_cust, cust_account), (account, cust_account)⟩

Example 7 Genotypes.
$\mathcal{G}_1 = \langle 1, 1, 1, 1, 0, 1 \rangle$; $\mathcal{G}_2 = \langle 0, 0, 0, 0, 1, 0 \rangle$; $\mathcal{G}_3 = \langle 0, 1, 1, 0, 0, 0 \rangle$;
$\mathcal{G}_4 = \langle 1, 1, 1, 1, 0, 0 \rangle$; $\mathcal{G}_5 = \langle 1, 0, 0, 1, 0, 1 \rangle$.

Encoding refers to the process that transforms the phenotype of a solution into its genotype representation, and *decoding* transforms the genotype of the solution into its corresponding phenotype representation, as defined by Algorithms 3 and 4, respectively.

Example 8 illustrates the process that two solutions in their phenotype representations (e.g., $\mathcal{P}_2$ and $\mathcal{P}_3$ in Example 5) evolve into two new solutions also in the phenotype representations using the decoding and encoding algorithms (Algorithms 3 and 4) and the crossover operator (see Section 5.3).

Example 8 Encoding, Crossover and Decoding
$\mathcal{P}_2$={({overseas_cust}, {cust_account})}

Algorithm 3 Encoding(Phenotype $\mathcal{P}$, A Sequence of Chromosomes $\mathcal{C}$)

```
1: construct a sequence of binary values $\mathcal{G} = \langle x_1, ..., x_m \rangle$, where $x_i = 0$ $(i = 1, ..., m)$;
2: **for** each $ELR_i = (ES_i^s, ES_i^t) \in \mathcal{P}$ **do**
3:    obtain a set of entity pairs $\mathcal{E} = ES_i^s \times ES_i^t$;
4:    **for** each $\varepsilon \in \mathcal{E}$ **do**
5:       find an entity pair $c_i \in \mathcal{C}$, and $\varepsilon = c_i$;
6:       $x_i \leftarrow 1$;
7: **return** $\mathcal{G}$.
```

Algorithm 4 Decoding(Genotype $\mathcal{G}$, A Sequence of Chromosomes $\mathcal{C}$)

```
1: $\mathcal{C}' \leftarrow \emptyset$;
2: **for** each pair of entities $c_i = (S_a, T_b) \in \mathcal{C}$ **do**
3:    **if** $x_i = 1$, where $x_i \in \mathcal{G}$ **then**
4:       $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{c_i\}$;
5: Phenotype $\mathcal{P} \leftarrow \emptyset$;
6: **while** $\mathcal{C}' \neq \emptyset$ **do**
7:    get a pair of entities $c_i = (S_a, T_b) \in \mathcal{C}'$, and remove $c_i$ from $\mathcal{C}'$;
8:    $ELR \leftarrow (ES_s, ES_t)$, where $ES_s = \{S_a\}$ and $ES_t = \{T_b\}$;
9:    **while** there exists a pair of entities $c_j = (S_x, T_y) \in \mathcal{C}'$, and $S_x \in ES_s$ or $T_y \in ES_t$
      **do**
10:      $ES_s \leftarrow ES_s \cup \{S_x\}$ and $ES_t \leftarrow ES_t \cup \{T_y\}$, and remove $c_j$ from $\mathcal{C}'$;
11:   $\mathcal{P} \leftarrow \mathcal{P} \cup \{ELR\}$;
12: **return** $\mathcal{P}$.
```

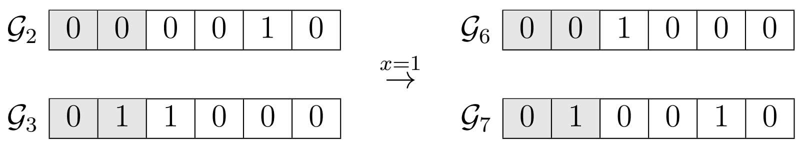
$\mathcal{P}_3$={({home_cust}, {cust_phone}), ({overseas_cust}, {customer})}

$\overset{encoding}{\Longrightarrow}$

$\mathcal{G}_2 = \langle 0, 0, 0, 0, 1, 0 \rangle$;

$\mathcal{G}_3 = \langle 0, 1, 1, 0, 0, 0 \rangle$;

$\overset{crossover}{\Longrightarrow}$


$\mathcal{G}_2$ 0 0 0 0 1 0
$\mathcal{G}_3$ 0 1 1 0 0 0
$\overset{x=1}{\rightarrow}$
$\mathcal{G}_6$ 0 0 1 0 0 0
$\mathcal{G}_7$ 0 1 0 0 1 0

$\mathcal{G}_6 = \langle 0, 0, 1, 0, 0, 0 \rangle$;

$\mathcal{G}_7 = \langle 0, 1, 0, 0, 1, 0 \rangle$;

$\overset{decoding}{\Longrightarrow}$

$\mathcal{P}_6$={({home_cust}, {cust_phone})}

$\mathcal{P}_7$={({overseas_cust}, {customer, cust_account})}

After applying mutation and crossover operators on solutions in genotype representations, not every newly generated solution (also in the genotype representation) can be decoded into a feasible phenotype representation, as they may not satisfy the conditions required in Definition 3. The part of the search space that is composed of such genotypes is usually called the infeasible space [ES03]. For example, genotype $\mathcal{G}_8 = \langle 0,0,0,1,1,1 \rangle$ cannot be decoded into a feasible phenotype solution, as there is no match between entities account and cust_phone, and as such {({overseas_cust, account}, {cust_phone, cust_account})} is not a feasible phenotype. Thus, we usually continue search for neighbours of the genotype whose decoded phenotype is not feasible until we find a genotype that can be decoded into a feasible phenotype.

5.5 Objective Function

As required by a genetic algorithm, the objective function models requirements of the problem to be solved and is responsible for calculating the fitness value of a solution representing its relative quality among the search space. A solution assigned the greatest fitness value during the search process is considered as an optimal solution and is returned. In principle, we expect the objective function that calculates the fitness value of a set of *ELRs* to model the following requirements:

1. It should assign a reasonable similarity to a single *ELR*. We propose this requirement because attributes of different entities (e.g., overseas_cust.name in RDB1 and cust_account.account_name in RDB2) are matched frequently due to coincidental overlap between their names or instances, as stated in Chapter 4. When a phenotype (e.g., $\mathcal{P}_2$ in Example 5) associates different entities (e.g., overseas_cust and cust_account), the similarity of the entities would be incorrectly increased due to the coincidental overlap between their attributes. Sometimes it is difficult to differentiate between the similarity of two equivalent (sets of) entities and the similarity of two different (sets of) entities that coincidentally have overlapping information. Therefore, we expect that the objective function could assign distinguishable similarities between equivalent entities and different entities.

2. The similarity of a single *ELR* associating equivalent n-to-m entities should be higher than the similarity of another *ELR* associating its subset entities, e.g., (n-1)-to-m entities. In Example 5, the *ELR* in $\mathcal{P}_1$ that associates {home_cust, overseas_cust} and {customer, cust_phone} should have higher similarity than the *ELR* in $\mathcal{P}_3$ that associates home_cust and cust_phone.

3. Given a single *ELR* that associates n-to-m entities, the objective function should be able to establish the partitioning of each entity set, in addition to the similarity of the *ELR*. For example, the objective function should identify that {home_cust, overseas_cust} and {customer, cust_phone} associated by phenotype $\mathcal{P}_1$ are horizontally and vertically partitioned, respectively.

4. Without using an absolute threshold to select *ELRs* and without knowing the number of *ELRs* to be returned, the objective function should assign the top fitness value to a solution (i.e., the returned solution) that contains as many *ELRs* whose similarities are relatively high as possible. For example, we assume that ELR_1 and ELR_2 in Example 2 have higher similarities than any other possible *ELRs*. Given phenotypes $\mathcal{P}_1 = \{ELR_1, ELR_2\}$ and $\mathcal{P}_4 = \{ELR_1\}$ in Example 5, $\mathcal{P}_1$ consists of two *ELRs* whose similarities are relatively high, but $\mathcal{P}_4$ is only composed of one of them. Thus, the objective function should assign a higher fitness value to $\mathcal{P}_1$ than to $\mathcal{P}_4$.

Given a solution (i.e., a set of *ELRs*), we first use the intuition behind the vector space model [SWY75] to calculate the similarity of each *ELR*, and then aggregate these similarities as the fitness value of the solution. In the remainder of this section, we first introduce the vector space model in Section 5.5.1, followed by an overview of the objective function in Section 5.5.2. We then present the technique for calculating *ELR* similarity in Sections 5.5.3 to 5.5.6 and the aggregation function in Section 5.5.7.

5.5.1 Background: vector space model

In traditional information retrieval [BYRN99], each document is described by a set of distinct words known as **index terms**. Words, such as articles and prepositions, that do not convey real world meaning are removed from the document. Each index term represents a word (probably occurring many times) in the document. For example, in a document that discusses the iPad, all occurrences of *iPad* are represented as a single index term. Similarly, a query posed over documents is also represented by index terms.

Generally speaking, the vector space model [SWY75] considers both documents and queries as **vectors** and calculates the **vector similarity** of the query and each document, as presented in Algorithm 5. An information retrieval system then ranks documents based on their similarities with the query and returns the top documents. To calculate the similarity of an *ELR*, we only need to appeal to the intuition behind the method for representing documents and the query as vectors and the method for calculating similarity of the query and each document. The process for ranking documents is not of interest to us here because of the evolutionary search approach we have adopted.

Algorithm 5 Vector Space Model (Documents $d_1, ..., d_n$, Query q)

```
1: obtain a set of index terms $K = \{k_1, ..., k_t\}$ that cover meaningful words in documents $d_1, ..., d_n$ and in the query $q$;
2: represent the query $q$ as a vector $\vec{q} = (w_{1q}, w_{2q}, ..., w_{tq})$, where,
3:     $w_{iq} > 0$ $(i = 1, ..., t)$ indicates that term $k_i \in K$ appears in query $\vec{q}$;
4:     $w_{iq} = 0$ $(i = 1, ..., t)$ indicates that term $k_i \in K$ does not appear in query $\vec{q}$;
5: for each document $d_j$ (j=1,...,n) do
6:     represent $d_j$ as a vector $\vec{d_j} = (w_{1j}, w_{2j}, ..., w_{tj})$;
7:     calculate the vector similarity of $\vec{q}$ and $\vec{d_j}$.
```

The first step in an information retrieval system is to collect a vocabulary of distinct words (Line *1* in Algorithm 5) known as index terms. The index terms are then used to describe the query q as vector $\vec{q}$ (Lines *2* to *4* in Algorithm 5) and each document d_j as vector $\vec{d_j}$ (Line *6* in Algorithm 5). In particular, the weight w_{ij}, best known as term frequency-inverse document frequency, of the index term $k_i \in K$ that appears in the document d_j can be calculated as $tf \times idf$, with $tf = \frac{t_i}{T}$ and $idf = log\frac{N}{n_i}$, where,

- t_i refers to the number of times term k_i appears in document $\vec{d_j}$;
- T denotes the total number of terms in $\vec{d_j}$;
- N represents the total number of documents used to build the vocabulary; and
- n_i is the number of documents in which term k_i appears.

The intuition behind $tf \times idf$ is that the more frequently term k_i appears in document $\vec{d_j}$, the more likely it is that it contributes to characterizing the meaning of the document, and thus tf is assigned a higher value; whilst the more often term k_i appears in different documents, the lesser its ability to distinguish document $\vec{d_j}$ from other documents is, and as such idf has a lower value. The weight of an index term in a vector indicates the importance of the term to describing the vector.

One of the methods for quantifying the similarity of each document d_j and the query q is to calculate the cosine of the angle between vectors $\vec{d_j}$ and $\vec{q}$, as presented in Function 5.1.

$$sim(\vec{d_j}, \vec{q}) = \frac{\sum_{i=1}^{t} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{t} (w_{ij})^2} \times \sqrt{\sum_{i=1}^{t} (w_{iq})^2}} \tag{5.1}$$

Although a large number of alternative methods can be used for calculating the similarity of a document and a query, the vector space model is known for its simplicity and good performance. The vector space model is either superior to or almost as good as the known alternatives [SWY75]. Assuming there is a query q, one of whose words (e.g., k_i) also appears in the document d_j, the weights w_{ij} and w_{iq} of the word k_i in both $\vec{d_j}$ and $\vec{q}$ would be greater than 0. A higher weight w_{ij} would derive a higher similarity of $\vec{d_j}$ and $\vec{q}$, which reflects that when a term is more unique to a document, its term weight in the document is higher, and therefore, when a query contains such a term, the document is the more relevant to the query. We use this intuition in calculating *ELR* similarities in the next sections.

5.5.2 Overview of the objective function

Following the intuition behind the vector space model, the objective function uses the following steps to calculate the fitness value for a phenotype, as presented in Algorithm 6:

- The first step (Lines *1* and *2*) is to construct a vocabulary of index terms for source and target schemas $\mathcal{S}$ and $\mathcal{T}$, respectively. In the context of inferring schematic correspondences, we consider each entity construct as a single index term. We also consider a set of attribute constructs belonging to different entities in a schema but representing the same notion as a single index term. For example, in RDB1 (Example 1), we consider the set of attribute constructs home_cust.name, overseas_cust.name and account.name as a single index term. The details are described in Section 5.5.3.

Algorithm 6 Objective Function(Source Schema $\mathcal{S}$, Target Schema $\mathcal{T}$, Phenotype $\mathcal{P} = \{ELR_1, ..., ELR_n\}$)

```
 1: construct a set of index terms $K^s$ for $\mathcal{S}$;
 2: construct a set of index terms $K^t$ for $\mathcal{T}$;
 3: for each $ELR_i = (ES_i^s, ES_i^t) \in \mathcal{P}$ do
 4:     represent the entity set $ES_i^s$ as a source vector $\vec{V_i^s}$;
 5:     represent the entity set $ES_i^t$ as a target vector $\vec{V_i^t}$;
 6: for each source entity $S_j$ that is not associated by $\mathcal{P}$ (i.e., $S_j \notin \cup_{i=1}^n ES_i^s$) do
 7:     represent the entity $S_j$ as a source vector $\vec{V_j^s}$
 8: for each target entity $T_j$ that is not associated by $\mathcal{P}$ (i.e., $T_j \notin \cup_{i=1}^n ES_i^t$) do
 9:     represent the entity $T_j$ as a target vector $\vec{V_j^t}$
10: for each $ELR_i = (ES_i^s, ES_i^t) \in \mathcal{P}$ do
11:     derive weights for index terms in vector $\vec{V_i^s}$ using $K^s$ and all vectors in $\mathcal{S}$;
12:     derive weights for index terms in vector $\vec{V_i^t}$ using $K^t$ and all vectors in $\mathcal{T}$;
13:     calculate the similarity $sim_i$ of vectors $\vec{V_i^s}$ and $\vec{V_i^t}$ using a cosine function;
14: given $sim_1$,...,$sim_n$, calculate the overall fitness value $f$ for phenotype $\mathcal{P}$;
15: return $f$.
```

- Given a phenotype $\mathcal{P}$, the second step (Lines *3* to *9*) represents the source and target schemas $\mathcal{S}$ and $\mathcal{T}$ as two vector spaces, where each *associated entity set* of $\mathcal{P}$ and each *unassociated entity* of $\mathcal{P}$ are represented as vectors. Each *associated entity set* of $\mathcal{P}$ with cardinality>1 is further represented as horizontal and vertical vectors, respectively, aiming to establish its partitioning (i.e., Requirement *3*). This step is presented in Section 5.5.4.

- The third step (Lines *11* to *12*) utilizes the $tf \times idf$ function to calculate term weights of vectors. In information retrieval, a term weight captures information about the frequency of appearance of a term in its vector using the *tf* part, and the more frequently it appears the higher of the *tf* part would be; it also captures the appearance of a term in other vectors using the *idf* part, where a higher frequency leads to a lower *idf* part.

  Here, if a construct is more unique to its entity, the construct is more likely to describe the entity. Thus, if a unique construct in schema $\mathcal{S}$ (e.g., account.balance in RDB1) is matched to a construct in schema $\mathcal{T}$ (e.g., cust_account.account_balance in RDB2), we put a higher weight on their similarity, thus giving rise to a higher similarity score of their entities (e.g., account and cust_account). In contrast, if a construct (e.g., attribute name) appears in several entities (e.g., home_cust, overseas_cust and account in RDB1), its ability to distinguish these entities is low, and as such a lower weight should be assigned to its similarity with a construct in the other schema (e.g., cust_account.account_name in RDB2). If an *ELR* associates different entities (e.g., overseas_cust and cust_account) that coincidentally have similar attributes, the similarity of the *ELR* should be low. This intuition helps to achieve Requirements *1* and *2*, as presented in Section 5.5.5.

- The fourth step (Line *13*) adopts Function 5.1 to calculate the similarity score of an *ELR* in the phenotype $\mathcal{P}$ by calculating the cosine of the angle between two vectors that represent the two *associated entity sets* of the *ELR*. Given an n-to-m *ELR*, each of its entity set is represented as horizontal and vertical vectors, respectively. Thus, this step computes four similarities between source horizontal and target horizontal vectors, source horizontal and target vertical vectors, source vertical and target horizontal vectors, and source vertical and target vertical vectors. The pair of partitioning types whose corresponding vector similarity is the maximum among the four similarities is identified as the *ELR*'s partitioning type. The detail is described in Section 5.5.6.

- The fifth step (Line *14*) presents a function that aggregates the *ELR* similarities into the fitness value of the phenotype. As stated in Requirement *4* in the preamble of Section 5.5, the top phenotype should contain as many *ELRs* that have relatively high similarities as possible, as presented in Section 5.5.7.

5.5.3 Identification of equivalent attributes

In this section, we illustrate the first step in Algorithm 6, where we build index terms for schemas $\mathcal{S}$ and $\mathcal{T}$, respectively. It maps to the process of constructing index terms from a space of documents in traditional information retrieval.

An index term can be an entity, an attribute or a set of attributes. In particular, when we consider a set of attributes as a single index term, it is usually the case that the set of attributes in different entities are equivalent. Note that we only count attributes belonging to distinct entities in a schema as equivalent attributes, as those attributes appearing in the same entity often represent different features of the entity. For example, in RDB1, we consider home_cust.name, overseas_cust.name and account.name as being the same.

We utilize a set of derived matches $\mathcal{M}$ between schemas $\mathcal{S}$ and $\mathcal{T}$ (defined in Section 5.4) to structure equivalent attributes into sets $\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_X$ in $\mathcal{S}$ (we repeat the same process for $\mathcal{T}$), where X varies given different schemas and is unknown before the identification process. This is because we require that source attributes represented as a term are not only similar to each other but also similar to a term in the target schema, and vice versa, because our overall aim is to use equivalent source and target terms to identify the similarity of two schemas. Furthermore, there may not be enough evidence to support the identification of equivalent attributes in a schema by matching the schema against itself. For example, equivalent attributes in distinct entities of a schema that are horizontally partitioned usually have disjoint instances, and thus the evidence for identifying them as being the same is weak. However, each of the same, e.g., source, attributes may have overlapping instances with a, e.g., target, attribute, which may provide more evidence to support us in identifying the source attributes as being the same.

Assuming $\mathcal{S}$ has entities $S_1, S_2, ..., S_\mu$, we denote the set of all their attributes as $\mathcal{L}$. We require that $\mathcal{L}_1 \cup \mathcal{L}_2 \cup ... \cup \mathcal{L}_X = \mathcal{L}$, and $\mathcal{L}_1 \cap \mathcal{L}_2 \cap ... \cap \mathcal{L}_X = \emptyset$. The cardinality of $\mathcal{L}_i$ $(i = 1, ..., X)$ belongs to $[1, \mu]$, which means that $\mathcal{L}_i$ contains at least an attribute that is different from any other attributes in $\mathcal{L}$, or at most has μ equivalent attributes, each of which belongs to a distinct entity.

Assume that sets of equivalent attributes $\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_j$ have been identified, and $\mathcal{L}' = \mathcal{L} - (\mathcal{L}_1 \cup \mathcal{L}_2 \cup ... \cup \mathcal{L}_j) \neq \emptyset$. We illustrate the method for identifying a new set $\mathcal{L}_{j+1} \subseteq \mathcal{L}'$ in the following text. In the running example, assuming sets $\mathcal{L}_1$ to $\mathcal{L}_4$ have been identified for RDB1, we further identify a new set of equivalent attributes $\mathcal{L}_5$ from $\mathcal{L}'$, as shown in Example 9.

Example 9 Equivalent attributes and remaining attributes in RDB1

$\mathcal{L}_1$={home_cust.id, overseas_cust.id}

$\mathcal{L}_2$={home_cust.name, overseas_cust.name, account.name}

$\mathcal{L}_3$={home_cust.birth, overseas_cust.birth}

$\mathcal{L}_4$={home_cust.a_id, overseas_cust.a_id, account.id}

$\mathcal{L}'$={home_cust.p_city, home_cust.p_area, ..., account.balance, account.tax}

The method for identifying a set of equivalent attributes from $\mathcal{L}'$ is presented in Algorithm 7. Given all attributes $\mathcal{L}$ in a schema, this algorithm will be used iteratively to retrieve all sets of equivalent attributes.

Algorithm 7 EquivalentAttributes(Attribute Set $\mathcal{L}'$, Derived Match Set $\mathcal{M}$)

```
1: $A \leftarrow$ the first attribute in $\mathcal{L}'$; // *to identify equivalent attributes of A.*
   // *Lines 2 to 8: identify sets of attributes that are potentially equivalent to A.*
2: identify attributes $B_1, ..., B_n$ to which $A$ is associated by matches in $\mathcal{M}$;
3: identify $\mathcal{A}_1$, ..., $\mathcal{A}_n$ sets of attributes using $\mathcal{M}$, where $\mathcal{A}_i = \{A, A_{i1}, ..., A_{im}\}$ ($1 \leqslant i \leqslant n$) satisfies:
4:     (i) $A \in \mathcal{A}_i$ and $|\mathcal{A}_i| \geqslant 1$;
5:     (ii) $\mathcal{A}_i \subseteq \mathcal{L}'$;
6:     (iii) all attributes of $\mathcal{A}_i$ are associated with the same attribute $B_i$ by matches in $\mathcal{M}$;
7:     (iv) each attribute in $\mathcal{A}_i$ belongs to a distinct entity;
8:     (v) $A_{ij}$ ($1 \leqslant j \leqslant m$) has higher similarity with $B_i$ than other attributes in the same entity.
   // *Lines 9 to 11: calculate similarity of each potential set and A.*
9: **for** each pair of $(\mathcal{A}_i, B_i)$ ($1 \leqslant i \leqslant n$) **do**
10:     $\mathcal{D}_i = \{d, d_{i1}, ..., d_{im}\} \leftarrow$ AttributeSim($\mathcal{A}_i$, $B_i$, $\mathcal{M}$);
11:     $avg(\mathcal{D}_i) \leftarrow \frac{\sum_{j=1}^{m} d_{ij}}{m}$;
   // *Line 12: select a set whose attributes are more similar to A than other sets.*
12: $(\mathcal{A}, B) \leftarrow$ one of $(\mathcal{A}_i, B_i)$, $1 \leqslant i \leqslant n$, that has the maximum $avg(\mathcal{D}_i)$;
   // *Lines 13 to 19: identify equivalent attributes of A from the selected set.*
13: $\mathcal{A}' \leftarrow \mathcal{A} - \{A\}$;
14: $\mathcal{C}'_1, ..., \mathcal{C}'_k \leftarrow$all combinatorial combinations of $\mathcal{A}'$ with all possible lengths;
15: $\mathcal{C}_0 \leftarrow \{A\}$, $\mathcal{C}_1 \leftarrow \mathcal{C}'_1 \cup \{A\}$, ..., $\mathcal{C}_k \leftarrow \mathcal{C}'_k \cup \{A\}$
16: **for** each of $\mathcal{C}_i$ ($0 \leqslant i \leqslant k$) **do**
17:     $\mathcal{D}^c_i = \{d, d_{i1}, ..., d_{im}\} \leftarrow$ AttributeSim($\mathcal{C}_i$, $B$, $\mathcal{M}$);
18:     $agg(\mathcal{D}^c_i) = \frac{(d+\sum_{j=1}^{m} d_{ij})^2}{|\mathcal{D}^c_i|}$;
19: **return** $\mathcal{C}_j$ whose $agg(\mathcal{D}^c_j)$ is the maximum among $agg(\mathcal{D}^c_1), ..., agg(\mathcal{D}^c_k)$.
```

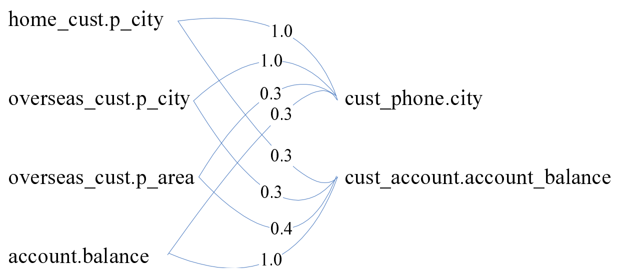


Figure 5.4: Matches between attributes of RDB1 and RDB2.

In the running example, Algorithm 7 takes as input attribute set $\mathcal{L}'$ and the set of derived matches $\mathcal{M}$ between attributes in $\mathcal{L}'$ and attributes in RDB2 (we show a subset of $\mathcal{M}$ in Figure 5.4). We start with the source attribute home_cust.p_city in $\mathcal{L}'$ (Line *1* in Algorithm 7), and find target attributes B_1 (cust_phone.city) and B_2 (cust_account.account_balance) to which home_cust.p_city is matched (Line *2* in Algorithm 7). Based on B_1 and B_2, two sets of source attributes $\mathcal{A}_1$ and $\mathcal{A}_2$ that contain potential equivalent attributes as A are identified (Lines *3* to *8* in

Algorithm 7) as:

$\mathcal{A}_1$={home_cust.p_city, overseas_cust.p_city, account.balance},

$\mathcal{A}_2$={home_cust.p_city, overseas_cust.p_area, account.balance}.

We apply Algorithm 8 to calculate the similarity of each candidate attribute set (e.g., $\mathcal{A}_1$ and $\mathcal{A}_2$) and attribute A (e.g., home_cust.p_city) (Lines *9* to *11* in Algorithm 7), thus enabling us to select a set (e.g., $\mathcal{A}_1$) whose attributes are more similar to attribute A than other sets (Line *12* in Algorithm 7). Finally, we identify a (sub)set of attributes from $\mathcal{A}_1$ as equivalent attributes as A (Lines *13* and *19* in Algorithm 7). Specifically, we explain Line *10* (Algorithm 8) and Lines *11* to *19* using the running example in the following text, respectively.

In Algorithm 8, remember that η (Line *1* Algorithm 8) denotes the number of different matchers we use to produce the input matches. As the maximum match similarity produced by each individual matcher is 1.0, η is also the maximum match similarity produced by all matchers we use, which denotes that two attributes are exactly the same, as defined in Definition 2 in Section 5.4. In the running example, we assume there is a single matcher used to produce the input matches, and thus $\eta = 1.0$. Using matches in Figure 5.4, for a candidate set, e.g., $\mathcal{A}_1$, we obtain the similarity of A (home_cust.p_city) and B_1 (cust_phone.city) as 1.0 (Line *2* in Algorithm 8), and similarities of other attributes in $\mathcal{A}_1$ and B_1 (Line *4* in Algorithm 8):

A_{11} (overseas_cust.p_city) and B_1 (cust_phone.city) as 1.0,

A_{12} (account.balance) and B_1 (cust_phone.city) as 0.3.

Similarly, we obtain the following similarities for set $\mathcal{A}_2$:

A (home_cust.p_city) and B_2 (cust_account.account_balance) as 0.3,

A_{21} (overseas_cust.p_area) and B_2 (cust_account.account_balance) as 0.4,

A_{22} (account.balance) and B_2 (cust_account.account_balance) as 1.0.

We can only infer similarity d_{ij} of attribute $A_{ij} \in \mathcal{A}_i$ and $A \in \mathcal{A}_i$ (Lines *5* to *8* in Algorithm 8) in the minimum case. Both attributes A (home_cust.p_city) and A_{11} (overseas_cust.p_city) are matched to B_1 (cust_phone.city) with the similarity of 1.0, and thus A, A_{11} and B_1 are exactly the same d_{11}=1.0. Given that A and B_1 are the same and the similarity of A_{12} (account.balance) and B_1 is 0.3, similarity d_{12} of A and A_{12} is 0.3 for sure (therefore, $\Delta_{AB_1} + \Delta_{A_{12}B_1} - \eta = 1.0 + 0.3 - 1.0 = 0.3$). However, given that A and A_{21} (overseas_cust.p_area) are matched to B_2 (cust_account.account_balance) with 0.3 and 0.4, respectively, A and A_{21} in the maximum case may be quite similar with the similarity of 0.9,

Algorithm 8 AttributeSim(Attribute Set $\mathcal{A}_i$, Attribute B_i, Derived Match Set $\mathcal{M}$)

```
1: $\eta \leftarrow$ the maximum match similarity;
2: $\Delta_{AB_i} \leftarrow$ the derived match similarity of attributes $A \in \mathcal{A}_i$ and $B_i$;
3: for each $A_{ij} \in \mathcal{A}_i$ $(1 \leqslant j \leqslant m)$ do
4:     $\Delta_{A_{ij}B_i} \leftarrow$ the derived match similarity of attributes $A_{ij}$ and $B_i$;
5:     if $\Delta_{AB_i} + \Delta_{A_{ij}B_i} - \eta \geqslant 0$ then
6:         $d_{ij} \leftarrow \Delta_{AB_i} + \Delta_{A_{ij}B_i} - \eta$;
7:     else
8:         $d_{ij} \leftarrow 0.0$;
9: $d \leftarrow \eta$;
10: $\mathcal{D}_i \leftarrow \{d, d_{i1}, ..., d_{im}\}$ is the set of similarities between attributes in $\mathcal{A}_i$ and $A$;
11: return $\mathcal{D}_i$.
```

but may be completely different in the minimum case. Therefore, we design $d_{ij} = \Delta_{AB_i} + \Delta_{A_{ij}B_i} - \eta$ to denote the similarity of A and A_{ij} in the minimum case. If $\Delta_{AB_i} + \Delta_{A_{ij}B_i} - \eta < 0$, we consider that A_{ij} and A may be completely different, and as such we set $d_{ij} = 0.0$. $d = \eta$ refers to the fact that A and A are the same (Line *9* in Algorithm 8). Thus, we obtain $\mathcal{D}_1 = \{1.0, 1.0, 0.3\}$ and $\mathcal{D}_2 = \{1.0, 0.0, 0.3\}$ using Algorithm 8.

We explain the maximum and minimum cases of comparison using a simple example. Assume $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $B_2 = \{a, b, c, d, e, f, g, 7, 8, 9\}$, their string similarity is 0.3. Given $A_{21} = \{0, 1, 2, 3, 4, 5, g, 7, 8, 9\}$ (the maximum case), the similarity of A_{21} and B_2 is 0.4, the similarity of A and A_{21} is 0.9. Given $A_{21} = \{a, b, c, d, z, y, x, w, v, u\}$ (the minimum case), the similarity of A_{21} and B_2 is also 0.4, but the similarity of A and A_{21} is 0.0.

Using **Lines *11* to *19* in Algorithm 7**, we consider attributes in set $\mathcal{A}_1$ to be more similar to A than set $\mathcal{A}_2$ (Lines *11* and *12* in Algorithm 7). However, not all attributes in $\mathcal{A}_1$ are the same as A, e.g., account.balance. Therefore, we need further steps to enumerate potential sets of equivalent attributes as A from set $\mathcal{A}_1$ (Lines *13* to *15* in Algorithm 7), calculate their similarities with A (Line *17* in Algorithm 7), and choose a set whose attributes are mostly similar to A as the result $\mathcal{L}_5$ (Lines *18* to *19* in Algorithm 7). Thus, we obtain $\mathcal{C}_0$, ..., $\mathcal{C}_3$ and their corresponding $\mathcal{D}^c_0$, ..., $\mathcal{D}^c_3$ as:

$\mathcal{C}_0$={home_cust.p_city}, $\mathcal{D}^c_0 = \{1.0\}$
$\mathcal{C}_1$={home_cust.p_city, overseas_cust.p_city}, $\mathcal{D}^c_1 = \{1.0, 1.0\}$
$\mathcal{C}_2$={home_cust.p_city, account.balance}, $\mathcal{D}^c_2 = \{1.0, 0.3\}$

$\mathcal{C}_3$={home_cust.p_city, overseas_cust.p_city, account.balance}, $\mathcal{D}^c_3 = \{1.0, 1.0, 0.3\}$

We derive $agg(\mathcal{D}^c_0) = 1.0$, $agg(\mathcal{D}^c_1) = 2.0$, $agg(\mathcal{D}^c_2) = 0.845$ and $agg(\mathcal{D}^c_3) = 1.76$ using the aggregation function in Line *18* in Algorithm 7. This aggregation value is high if similarities in $\mathcal{D}^c_i$ are close to the highest similarity (i.e., η), which means that attributes in C_i are similar to attribute A. We then obtain attributes in $\mathcal{C}_1$ as being the same, and thus $\mathcal{L}_5$ = {home_cust.p_city, overseas_cust.p_city}. For the convenience of future explanation, we extend equivalent attributes in Example 9 to a full list of equivalent attributes in RDB1 and in RDB2 in Example 10, where we omit attribute sets whose cardinality is 1 (attributes that are different from others) for the sake of simplicity.

Example 10 Equivalent attributes in RDB1 and in RDB2.
In RDB1:
$\mathcal{L}^s_1$={home_cust.id, overseas_cust.id}
$\mathcal{L}^s_2$={home_cust.name, overseas_cust.name, account.name}
$\mathcal{L}^s_3$={home_cust.birth, overseas_cust.birth}
$\mathcal{L}^s_4$={home_cust.a_id, overseas_cust.a_id, account.id}
$\mathcal{L}^s_5$={home_cust.p_city, overseas_cust.p_city}
$\mathcal{L}^s_6$={home_cust.p_area, overseas_cust.p_area}
$\mathcal{L}^s_7$={home_cust.p_local, overseas_cust.p_local}
In RDB2:
$\mathcal{L}^t_1$={customer.key, cust_phone.key}
$\mathcal{L}^t_2$={customer.account_key, cust_account.key}

The step of identifying equivalent attributes for the source (or target) schema allows us to structure source (or target) attributes that represent the same concept as a term, thus helping us to represent the source (or target) schema as a vector space, as presented in Section 5.5.4.

5.5.4 Source and target vector spaces

In this section, we discuss the second step of Algorithm 6 presented in Section 5.5.2, where given a phenotype $\mathcal{P} = \{ELR_1,\ ELR_2,\ ...,\ ELR_n\}$, the source and target schemas $\mathcal{S} = \{S_1, S_2, ..., S_\mu\}$ and $\mathcal{T} = \{T_1, T_2, ..., T_\upsilon\}$ are represented as source and target vector spaces $\mathbf{V_S}$ and $\mathbf{V_T}$, as defined in Definition 5.

Definition 5 ***Vector Space.*** *Given a phenotype $\mathcal{P}$, a schema $\mathcal{S}$ is represented as a* **vector space** $\mathbf{V} = \{\vec{V}_1, ..., \vec{V}_N\}$*, where each vector* $\vec{V}_i = (w_0 \cdot k_0, w_1 \cdot k_1, ..., w_\varphi \cdot k_\varphi)$ *represents either an associated entity set of $\mathcal{P}$ or an unassociated entity of $\mathcal{P}$. Specifically,* $w_j \cdot k_j$ $(j = 0, ..., \varphi)$ *represent* **dimensions** *in vector $\vec{V}_i$, where each k_j is a* **term** *representing a (set of) construct(s) and w_j is the* **term weight**.

Given the phenotype $\mathcal{P}$, we take the source vector space $\mathbf{V_S}$ representing schema $\mathcal{S}$ as an example. For each $ELR_i = (ES^s_i, ES^t_i) \in \mathcal{P}$ $(i = 1, ..., n)$ where ES^s_i represents an *associated entity set*, we use a vector to represent ES^s_i. For the remaining entities (e.g., S_j) in $\mathcal{S}$, which are the *unassociated entities* of $\mathcal{P}$ (i.e., $S_j \notin \cup^n_{i=1} ES^s_i$), we use separate source vectors to represent each of them (e.g., S_j). Thus, every entity in $\mathcal{S}$ is represented by or contained in a vector in the source vector space $\mathbf{V_S}$. We present source and target vector spaces constructed using phenotypes $\mathcal{P}_1$ and $\mathcal{P}_2$ (Example 5) in Example 11.

Example 11 ***Vector space.***
Using $\mathcal{P}_1$, source and target vector spaces $\mathbf{V_{S1}}$ and $\mathbf{V_{T1}}$ are constructed as:
$\mathbf{V_{S1}} = \{\vec{V}^s_1, \vec{V}^s_2\}$*, where $\vec{V}^s_1$ and $\vec{V}^s_2$ represent the associated entity sets* {home_cust, overseas_cust} *and* {account}*, respectively;*
$\mathbf{V_{T1}} = \{\vec{V}^t_1, \vec{V}^t_2\}$*, where $\vec{V}^t_1$ and $\vec{V}^t_2$ represent the associated entity sets* {customer, cust_phone} *and* {cust_account}*, respectively.*

Using $\mathcal{P}_2$, source and target vector spaces $\mathbf{V_{S2}}$ and $\mathbf{V_{T2}}$ are constructed as:
$\mathbf{V_{S2}} = \{\vec{V}^s_1, \vec{V}^s_2, \vec{V}^s_3\}$*, where $\vec{V}^s_1$ represents the associated entity set* {overseas_cust}*, and $\vec{V}^s_2$ and $\vec{V}^s_3$ represent the unassociated entities* home_cust *and* account*;*
$\mathbf{V_{T2}} = \{\vec{V}^t_1, \vec{V}^t_2, \vec{V}^t_3\}$*, where $\vec{V}^t_1$ represents the associated entity set* {cust_account}*, and $\vec{V}^t_2$ and $\vec{V}^t_3$ represents the unassociated entities* customer *and* cust_phone.

A single vector, as defined in Definition 6, in a vector space represents either an *associated entity set* (with cardinality=1) of the phenotype $\mathcal{P}$ or an *unassociated entity* of $\mathcal{P}$.

Definition 6 ***Single Vector.*** *An entity S_i, containing attributes $S_i.A_1, ..., S_i.A_\alpha$, is represented as a* **single vector** $\vec{V}_i = (w_0 \cdot k_0, w_1 \cdot k_1, ..., w_\alpha \cdot k_\alpha)$*, where $k_0 = S_i$ is the term for the entity construct and $k_j = S_i.A_j$ $(j = 1, ..., \alpha)$ is a term for each attribute construct $S_i.A_j$.*

Following on with Example 11, using $\mathcal{P}_2$, the *associated entity set* {cust_account} is represented as a single vector $\vec{V}_1^t \in \mathbf{V_{T2}}$, and the *unassociated entity* account is also represented as a single vector $\vec{V}_3^s \in \mathbf{V_{S2}}$, as presented in Example 12.

Example 12 ***Single vector.***
$\vec{V}_3^s = (w_0 \cdot \text{account}, w_1 \cdot \text{account.id}, w_2 \cdot \text{account.name}, w_3 \cdot \text{account.balance},$
$w_4 \cdot \text{account.tax});$
$\vec{V}_1^t = (w_0 \cdot \text{cust\_account}, w_1 \cdot \text{cust\_account.key}, w_2 \cdot \text{cust\_account.account\_name},$
$w_3 \cdot \text{cust\_account.account\_balance}).$

If a vector represents an *associated entity set* of the phenotype $\mathcal{P}$ whose cardinality>1, it is further extended into a horizontal vector (Definition 7) and a vertical vector (Definition 8). This enables the objective function to establish the partitioning type of the *associated entity set.*

The basic idea of *horizontal partitioning* is that an original entity is partitioned along its instances into new entities, and as such all attributes of the original entity are present in each of new entities. Given an *associated entity set* whose cardinality>1, a horizontal vector models the requirement for identifying the entity set as being horizontally partitioned, which is that the more attributes are shared by all the entities in the set, the more likely the entities are horizontally partitioned. A set of attributes shared by entities are the *equivalent attributes* of the entities, as identified in Section 5.5.3. To support the above notion within the vector space model, the horizontal vector is defined in Definition 7.

Definition 7 ***Horizontal Vector.*** *A set of entities $\{S_1, ..., S_\varrho\}$ is expressed as a* **horizontal vector** $\vec{V}^H = (w_0 \cdot k_0,\ w_1 \cdot k_1,\ ...,\ w_P \cdot k_P)$*, where $k_0 = \{S_1, ..., S_\varrho\}$ is the term representing the set of entity constructs and k_i $(i = 1, ..., P)$ are terms representing attribute constructs. For equivalent attributes of all entities $S_1.A_c, ..., S_\varrho.A_c$, k_i is defined as $\{S_1.A_c, ..., S_\varrho.A_c\}$; for equivalent attributes of a subset of all entities, e.g., $S_1.A_{nc}$ and $S_2.A_{nc}$, k_i is defined as $\{S_1.A_{nc}, S_2.A_{nc}, S_3.\psi, ..., S_\varrho.\psi\}$, where $S_j.\psi$ $(j = 3, ..., \varrho)$ represents that the entity S_j does not contain an equivalent attribute as entities S_1 and S_2.*

Following on with Example 11, $\vec{V}_1^s \in \mathbf{V_{S1}}$ and $\vec{V}_1^t \in \mathbf{V_{T1}}$ are further represented as horizontal vectors $\vec{V}_1^{sH}$ and $\vec{V}_1^{tH}$, as presented in Example 13.

Example 13 ***Horizontal vectors.***
$\vec{V}_1^{sH} = (w_0^{sH} \cdot \{\text{home\_cust, overseas\_cust}\}, w_1^{sH} \cdot \{\text{home\_cust.id, overseas\_cust.id}\},$

w_2^{sH}·{home_cust.name, overseas_cust.name},
w_3^{sH}·{home_cust.birth, overseas_cust.birth},
w_4^{sH}·{home_cust.a_id, overseas_cust.a_id},
w_5^{sH}·{home_cust.p_city, overseas_cust.p_city},
w_6^{sH}·{home_cust.p_area, overseas_cust.p_area},
w_7^{sH}·{home_cust.p_local, overseas_cust.p_local});
$\vec{V}_1^{tH} = (w_0^{tH}$·{customer, cust_phone}, w_1^{tH}·{customer.key, cust_phone.key},
w_2^{tH}·{customer.c_fname, cust_phone.ψ},
w_3^{tH}·{customer.c_lname, cust_phone.ψ},
w_4^{tH}·{customer.c_birth, cust_phone.ψ},
w_5^{tH}·{customer.account_key, cust_phone.ψ},
w_6^{tH}·{customer.ψ, cust_phone.city},
w_7^{tH}·{customer.ψ, cust_phone.area},
w_8^{tH}·{customer.ψ, cust_phone.local},
w_9^{tH}·{customer.ψ, cust_phone.extension});

In Example 13, $\vec{V}_1^{sH}$ and $\vec{V}_1^{tH}$ model the requirement for identifying the source entity set {home_cust, overseas_cust} and the target entity set {customer, cust_phone} as being horizontally partitioned, respectively. Equivalent attributes shared by all entities in the source set (e.g., home_cust.name, overseas_cust.name) are represented as a single dimension in $\vec{V}_1^{sH}$ (e.g., w_2^{sH}·{home_cust.name, overseas_cust.name}), which indicates that entities home_cust and overseas_cust share an equivalent attribute name. There are also attributes not shared by all entities in the target set (e.g., customer.c_birth), and thus each of such attributes and a ψ attribute are represented as a single dimension in $\vec{V}_1^{tH}$ (e.g., w_4^{tH}·{customer.c_birth, cust_phone.ψ}. If more dimensions in a horizontal vector do not contain ψ attributes (e.g., $\vec{V}_1^{sH}$), there are more attributes shared by all the entities in the set that the horizontal vector represents, and thus it is more likely that these entities are horizontally partitioned (e.g., {home_cust, overseas_cust}); otherwise the entities are not horizontally partitioned (e.g., {customer, cust_phone}).

The basic idea of *vertical partitioning* is that an original entity is partitioned into new entities whose attributes are subsets of the original's. In particular, some attributes are present in each new entity, referring to key attributes, whereas other attributes of the original entity are present only once across all the new entities. Similar to a horizontal vector, a vertical vector models the requirement for identifying that an *associated entity set* whose cardinality>1 is vertically partitioned, which is that some attributes are shared by all the entities in the set and the remaining attributes are not shared by all the entities. To support this intuition, the vertical vector is defined in Definition 8.

Definition 8 ***Vertical Vector.*** *A set of entities* $\{S_1, ..., S_\varrho\}$ *is expressed as a* **vertical vector** $\vec{V}^V$= $(w_0{\cdot}k_0, w_1{\cdot}k_1, ..., w_Q{\cdot}k_Q)$*, where* $k_0 = \{S_1, ..., S_\varrho\}$ *is the term representing the set of entity constructs and* k_i $(i = 1, ..., Q)$ *are terms representing attribute constructs. For equivalent attributes of all entities* $S_1.A_c, ..., S_\varrho.A_c$*,* k_i *is defined as* $\{S_1.A_c, ..., S_\varrho.A_c\}$ *referring to key attributes of all entities; for equivalent attributes of a subset of all entities, e.g.,* $S_1.A_{nc}$ *and* $S_2.A_{nc}$*,* k_i *is defined as* $\{S_1.A_{nc}\}$ *and* $k_{i'} = \{S_2.A_{nc}\}$ $(i \neq i')$*; for an attribute* A_{nc} *in* S_j $(j = 1, ..., \varrho)$ *that is different in entities* $S_1, ..., S_\varrho$*,* k_i *is defined as* $\{S_j.A_{nc}\}$*.*

Following on with Example 11, $\vec{V}_1^s \in \mathbf{V_{S1}}$ and $\vec{V}_1^t \in \mathbf{V_{T1}}$ are further represented as vertical vectors $\vec{V}_1^{sV}$ and $\vec{V}_1^{tV}$, as presented in Example 14.

Example 14 ***vertical vectors.***

$\vec{V}_1^{sV} = (w_0^{sV}{\cdot}\{\text{home\_cust, overseas\_cust}\}, w_1^{sV}{\cdot}\{\text{home\_cust.id, overseas\_cust.id}\},$
$w_2^{sV}{\cdot}\{\text{home\_cust.name, overseas\_cust.name}\},$
$w_3^{sV}{\cdot}\{\text{home\_cust.birth, overseas\_cust.birth}\},$
$w_4^{sV}{\cdot}\{\text{home\_cust.a\_id, overseas\_cust.a\_id}\},$
$w_5^{sV}{\cdot}\{\text{home\_cust.p\_city, overseas\_cust.p\_city}\},$
$w_6^{sV}{\cdot}\{\text{home\_cust.p\_area, overseas\_cust.p\_area}\},$
$w_7^{sV}{\cdot}\{\text{home\_cust.p\_local, overseas\_cust.p\_local}\});$
$\vec{V}_1^{tV} = (w_0^{tV}{\cdot}\{\text{customer, cust\_phone}\}, w_1^{tV}{\cdot}\{\text{customer.key, cust\_phone.key}\},$
$w_2^{tV}{\cdot}\{\text{customer.c\_fname}\}, w_3^{tV}{\cdot}\{\text{customer.c\_lname}\},$
$w_4^{tV}{\cdot}\{\text{customer.c\_birth}\}, w_5^{tV}{\cdot}\{\text{customer.account\_key}\},$
$w_6^{tV}{\cdot}\{\text{cust\_phone.city}\}, w_7^{tV}{\cdot}\{\text{cust\_phone.area}\},$
$w_8^{tV}{\cdot}\{\text{cust\_phone.local}\}, w_9^{tV}{\cdot}\{\text{cust\_phone.extension}\});$

$\vec{V}_1^{sV}$ and $\vec{V}_1^{tV}$ model the requirement for identifying the source entity set {home_cust, overseas_cust} and the target entity set {customer, cust_phone} as being vertically partitioned, respectively. A set of equivalent attributes shared by all the entities in the target set is represented as a single dimension in $\vec{V}_1^{tV}$ (e.g., $w_1^{tV}{\cdot}\{\text{customer.key, cust\_phone.key}\}$), referred to as key attributes. There are also attributes in the target set not shared by all the entities (e.g., customer.c_fname), which refer to attributes of the original entity that are present only once across all the new entities, and thus each of such attributes is represented as a single dimension (e.g., $w_2^{tV}\cdot$\{customer.c_fname\}). In the source set, equivalent attributes shared by all the entities are represented as a single dimension in $\vec{V}_1^{sV}$ (e.g., $w_2^{sV}\cdot$\{home_cust.name, overseas_cust.name\}). As there is not such a dimension in $\vec{V}_1^{sV}$ whose attribute is present only once in all the entities in the source set, the vertical partitioning requirement is not met. Thus, the source entity set is not vertically partitioned.

5.5.5 Weight calculation

In Section 5.5.3, we structure equivalent attributes in source schema $\mathcal{S}$ and target schema $\mathcal{T}$ into index terms, and use the index terms to construct source and target vector spaces $\mathbf{V_S}$ and $\mathbf{V_T}$ to represent source and target schemas $\mathcal{S}$ and $\mathcal{T}$, as presented in Section 5.5.4. In this section, we describe the method for calculating the term weights of each vector, which maps to the third step of the objective function presented in Algorithm 6 in Section 5.5.2. We follow the intuition behind the design of the vector space model, and use the $tf \times idf$ method to calculate weight w_i for term k_i ($i = 0, ..., \varphi$) in each vector $\vec{V} = (w_0 \cdot k_0, w_1 \cdot k_1, ..., w_\varphi \cdot k_\varphi) \in \mathbf{V_S}$ ($\mathbf{V_T}$), where $tf = \frac{t_i}{T}$ and $idf = log\frac{N}{n_i}$ are defined as follows:

- t_i refers to the number of the equivalent constructs (entities or attributes) in the term k_i (we consider ψ used in horizontal vectors as a construct as well);
- T denotes the total number of constructs in vector $\vec{V}$;
- N represents the total number of vectors in $\mathbf{V_S}$; and
- n_i denotes the number of vectors in $\mathbf{V_S}$ that have the equivalent constructs as the term k_i.

For example, the term weight w_1 for term \{home_cust.id, overseas_cust.id\} in $\vec{V}_1^s$ (e.g., $\vec{V}_1^{sH}$ in Example 13) $\in \mathbf{V_{S1}}$ (Example 11) can be calculated as: t_1=2 for the two attribute constructs in the term; T=16 for the total number of 16 constructs in $\vec{V}_1^{sH}$; N=2 as there are 2 vectors $\vec{V}_1^s$ (e.g., $\vec{V}_1^{sH}$) and $\vec{V}_2^s$ in $\mathbf{V_{S1}}$; n_1=2 as term account.id in $\vec{V}_2^s \in \mathbf{V_{S1}}$ contains equivalent construct as term \{home_cust.id, overseas_cust.id\} in $\vec{V}_1^s$ (i.e., $\vec{V}_1^{sH}$) $\in \mathbf{V_{S1}}$.

To meet Requirement *1* in the preamble of Section 5.5, the idea is that the more unique a term to a vector, the term is more likely to describe the vector.

Thus, if n_i of w_i is lower (the *idf* part of w_i is higher), the constructs in the term are more likely to represent the meaning of the vector. We take phenotype $\mathcal{P}_2$ as an example to explain the necessity of the n_i part in the term weight for this method. There are $\vec{V}_1^s$, $\vec{V}_2^s$ and $\vec{V}_3^s$ in $\mathbf{V_{S2}}$ (Example 11) representing the schema $\mathcal{S}$. The n_i part for the term overseas_cust.name in $\vec{V}_1^s$ is 3, because $\vec{V}_2^s$ and $\vec{V}_3^s$ have terms home_cust.name and account.name, respectively, which contain equivalent constructs as the term overseas_cust.name. In contrast, n_i of other terms in $\vec{V}_1^s$ (e.g., overseas_cust.birth) is 1, because the terms are unique to $\vec{V}_1^s$. Thus, n_i of 3 for the term overseas_cust.name in $\vec{V}_1^s$ gives rise to a lower *idf* value than other terms in $\vec{V}_1^s$. Therefore, even though entities overseas_cust ($\vec{V}_1^s$) and cust_account ($\vec{V}_1^t$) are associated by $\mathcal{P}_2$ and the coincidental match between overseas_cust.name and cust_account.account_name provides evidence to support their association, the lower weight assigned to the term overseas_cust.name could help to reduce this coincidental evidence.

To address Requirement *2*, the idea is that the more often a term appears in a single vector, the more likely it is able to represent the meaning of the document, and if so the *tf* part of the term weight should be assigned a higher value. We use phenotypes $\mathcal{P}_1$ and $\mathcal{P}_3$ as an example. In Example 11, $\vec{V}_1^s$ (e.g., $\vec{V}_1^{sH}$) represents {home_cust, overseas_cust} associated by $\mathcal{P}_1$, which are horizontally partitioned. The equivalent attributes (e.g., home_cust.p_city and overseas_cust.p_city) are structured as a term in $\vec{V}_1^{sH}$, thus giving rise to t_i=2 and n_i=1. In contrast, entities home_cust and overseas_cust are represented by two separate vectors given $\mathcal{P}_3$, and as such the weight of term home_cust.p_city in the vector home_cust are calculated using t_i=1 and n_i=2. Thus, the term {home_cust.p_city, overseas_cust.p_city} (given $\mathcal{P}_1$) has a higher *tf* and a lower *idf* than the term home_cust.p_city (given $\mathcal{P}_3$), and thus has a higher weight. The difference of weights helps to assign a higher similarity to equivalent entities {home_cust, overseas_cust} and {customer, cust_phone} associated by $\mathcal{P}_1$ than the pair of entities home_cust and cust_phone associated by $\mathcal{P}_3$.

5.5.6 Vector similarity

This section describes the fourth step of Algorithm 6 in Section 5.5.2, i.e., the function for calculating vector similarity. Given a phenotype $\mathcal{P}$, each *ELR* in $\mathcal{P}$ associates two (sets of) entities represented by a pair of vectors $\vec{V}^s$ and $\vec{V}^t$. We calculate the similarity of $\vec{V}^s$ and $\vec{V}^t$, and regard it as the *ELR* similarity.

In Example 11, between $\mathbf{V_{S1}}$ and $\mathbf{V_{T1}}$ (for phenotype $\mathcal{P}_1$) we calculate the similarity of $\vec{V}_1^s$ and $\vec{V}_1^t$ as the similarity of the *ELR* ({home_cust, overseas_cust}, {customer, cust_phone}), and calculate the similarity of $\vec{V}_2^s$ and $\vec{V}_2^t$ as the similarity of the *ELR* ({account}, {cust_account}). In particular, the similarity of $\vec{V}^s$ and $\vec{V}^t$ is derived as the maximum similarity among similarities of $\vec{V}_1^{sH}$ and $\vec{V}_1^{tH}$, $\vec{V}_1^{sH}$ and $\vec{V}_1^{tV}$, $\vec{V}_1^{sV}$ and $\vec{V}_1^{tH}$, and $\vec{V}_1^{sV}$ and $\vec{V}_1^{tV}$. We consider the fittest partitioning of the *ELR* as the corresponding partitioning of the two vectors assigned with the maximum vector similarity among the four similarities.

For phenotype $\mathcal{P}_2$, although each of vector spaces $\mathbf{V_{S2}}$ and $\mathbf{V_{T2}}$ has three vectors, as $\mathcal{P}_2$ only contains a single *ELR* that associates entities overseas_cust (represented as $\vec{V}_1^s \in \mathbf{V_{S2}}$) and cust_account (represented as $\vec{V}_1^t \in \mathbf{V_{T2}}$), we only need to calculate the similarity of $\vec{V}_1^s$ and $\vec{V}_1^t$ as the *ELR* similarity.

Let us assume that we have vectors $\vec{V}^s = (w_{s0} \cdot k_{s0}, w_{s1} \cdot k_{s1}, ..., w_{sM} \cdot k_{sM})$ and $\vec{V}^t = (w_{t0} \cdot k_{t0}, w_{t1} \cdot k_{t1}, ..., w_{tN} \cdot k_{tN})$. We adopt the traditional cosine function, presented in Function 5.1 in Section 5.5.1, to calculate their similarity.

We first identify 1-to-1 matched terms. As defined in Section 5.5.4, a term in a vector represents either a construct (i.e., an entity or an attribute) or a set of constructs. Thus, if there are matches between two (sets of) constructs represented by two terms, we consider that the two terms are 1-to-1 matched. We define the *term similarity* as the (average) match similarity of the two (sets of) constructs the two terms represent, which is a float value between $[0, \eta]$ (Recall that we assume there are η matchers used for producing the input matches and the maximum match similarity produced by each individual matcher is 1.0. Thus, η also represents the maximum similarity that denotes that two constructs are exactly the same). We use k_{s0} and k_{t0} to represent the two (sets of) entity constructs in $\vec{V}^s$ and $\vec{V}^t$, respectively. We assume that if term k_{si} of $\vec{V}^s$ $(0 \leqslant i \leqslant M)$ has a term similarity greater than 0 with more than one term in $\vec{V}^t$, the matched term of k_{si} is the term k_{tj} in $\vec{V}^t$ that has the maximum term similarity with k_{si}. We require that k_{tj} in $\vec{V}^t$ cannot be the matched term of any other term $k_{si'}$ in $\vec{V}^s$ $(0 \leqslant i' \leqslant M$ and $i' \neq i)$.

If a term contains ψ attributes in a horizontal vector (e.g., in Example 13, term {customer.ψ, cust_phone.city} in $\vec{V}_1^{tH}$), we consider the match similarity of the ψ attribute and any attribute as 0.0. Thus, the more ψ attributes that are contained in a term, the lower the term similarity between this term and its equivalent term (diluted by the match similarity of 0.0), and the lower the vector similarity (see

Function 5.2 below). Therefore, introducing ψ attributes to horizontal vectors is used to identify whether a set of entities is horizontally partitioned. For example, given a set of non-*HP* entities (e.g., {customer, cust_phone} in phenotype $\mathcal{P}_1$), several terms of its horizontal vector (e.g., $\vec{V}_1^{tH}$ in Example 13) may contain ψ attributes, and thus the term similarities would be low, which would also result in a low similarity of the horizontal vector (e.g., $\vec{V}_1^{tH}$) and the other vector (e.g., $\vec{V}_1^{sH}$ in Example 13 or $\vec{V}_1^{sV}$ in Example 14), thus indicating that the set of entities is not horizontally partitioned. By contrast, for a set of good *HP* entities (e.g., {home_cust, overseas_cust} in $\mathcal{P}_1$), its horizontal vector usually has few ψ attributes (e.g., $\vec{V}_1^{sH}$ in Example 13), thus giving rise to a high vector similarity. Therefore, we can infer that {home_cust, overseas_cust} is horizontally partitioned, and {customer, cust_phone} is vertically partitioned. In particular, we show matched terms between vectors $\vec{V}_1^{sH}$ (Example 13) and $\vec{V}_1^{tV}$ (Example 14) in Example 15.

Example 15 ***Matched terms.***

{home_cust, overseas_cust} $\Longleftrightarrow$ {customer, cust_phone}
{home_cust.id, overseas_cust.id} $\Longleftrightarrow$ {customer.key, cust_phone.key}
{home_cust.name, overseas_cust.name} $\Longleftrightarrow$ {customer.c_fname}
{home_cust.birth, overseas_cust.birth} $\Longleftrightarrow$ {customer.c_birth}
{home_cust.a_id, overseas_cust.a_id} $\Longleftrightarrow$ {customer.account_key}
{home_cust.p_city, overseas_cust.p_city} $\Longleftrightarrow$ {cust_phone.city}
{home_cust.p_area, overseas_cust.p_area} $\Longleftrightarrow$ {cust_phone.area}
{home_cust.p_local, overseas_cust.p_local} $\Longleftrightarrow$ {cust_phone.local}

Assume that we have identified R 1-to-1 matched terms between $\vec{V}^s$ and $\vec{V}^t$, where $R \leqslant min(M, N)$. For the sake of simplicity, we reallocate terms in vector $\vec{V}^t$, in order to express matched terms between $\vec{V}^s$ and $\vec{V}^t$ using the same subscript numbers. Thus, we represent the source and target vectors as:

$$\vec{V}^s = (w_{s0} \cdot k_{s0}, ..., w_{sR} \cdot k_{sR}, ..., w_{sM} \cdot k_{tM})$$

$$\vec{V}^t = (w'_{t0} \cdot k'_{t0}, ..., w'_{tR} \cdot k'_{tR}, ..., w'_{tN} \cdot k'_{tN})$$

In particular, $w_{sl} \cdot k_{sl}$ and $w'_{tl} \cdot k'_{tl}$ $(0 \leqslant l \leqslant R)$ are 1-to-1 matched terms between $\vec{V}^s$ and $\vec{V}^t$, whose term similarity we denote as m_l. We calculate the similarity of vectors $\vec{V}^s$ and $\vec{V}^t$ as:

$$sim(\vec{V}^s, \vec{V}^t) = \frac{\sum_{l=0}^{R} m_l \times w_l \times w'_l}{\sqrt{\sum_{i=0}^{M}(w_i)^2} \times \sqrt{\sum_{j=0}^{N}(w'_j)^2}} \tag{5.2}$$

For a phenotype $\mathcal{P} = \{ELR_1, ELR_2, ..., ELR_n\}$, where each $ELR_i \in \mathcal{P}$ associates entity sets ES_i^s and ES_i^t, we generate different types of vectors for ES_i^s and ES_i^t given their cardinalities, as defined in Section 5.5.4. If ES_i^s and ES_i^t are 1-to-1 entities, we construct a single vector to represent each of them. If ES_i^s and ES_i^t are n-to-m entities, we construct a horizontal vector $\vec{V}^{sH}$ and a vertical vector $\vec{V}^{sV}$ for ES_i^s, and similarly vectors $\vec{V}^{tH}$ and $\vec{V}^{tV}$ for ES_i^t.

In the case where ES_i^s and ES_i^t are 1-to-1 entities, we apply Function 5.2 to calculate their similarity. When ES_i^s and ES_i^t are n-to-m entities, we calculate four kinds of similarities using Function 5.2 for the following combinations of partitioning: *HP vs HP*, *HP vs VP*, *VP vs HP*, *VP vs VP*, i.e., $s_{H-H} = sim(\vec{V}^{sH}, \vec{V}^{tH})$, $s_{H-V} = sim(\vec{V}^{sH}, \vec{V}^{tV})$, $s_{V-H} = sim(\vec{V}^{sV}, \vec{V}^{tH})$ and $s_{V-V} = sim(\vec{V}^{sV}, \vec{V}^{tV})$ to denote the fitness of ES_i^s and ES_i^t that are *HP* and *HP*, *HP* and *VP*, *VP* and *HP*, and *VP* and *VP*, respectively. We consider the similarity of ES_i^s and ES_i^t as the maximum value of s_{H-H}, s_{H-V}, s_{V-H}, and s_{V-V}, and report the corresponding partitioning types.

5.5.7 Aggregation

Given a phenotype $\mathcal{P} = \{ELR_1, ELR_2, ..., ELR_n\}$, we calculate the similarity, denoted as s_i, of each $ELR_i \in \mathcal{P}$ $(i = 1, ..., n)$ in Sections 5.5.3 and 5.5.6. In this section, we illustrate the last step in computing a value for the objective function (Algorithm 6 in Section 5.5.2), which calculates the fitness value for the phenotype $\mathcal{P}$ by aggregating *ELR* similarities. As stated in Requirement *4* in the preamble of Section 5.5, we anticipate that the aggregation function assigns a higher value to a phenotype, if more *ELRs* that have comparatively high similarities are contained by the phenotype. Note that we do not define a similarity threshold that arbitrarily denotes whether the pair of entity sets associated by an *ELR* is similar or not, and we do not know in advance how many such *ELRs* should be contained in the phenotype.

The *sum* aggregation function tends to favor a larger number of aggregated values and is higher when more s_i values are aggregated, even though these values are low; whereas the *average* aggregation tends to favor the highest value among s_i $(i = 1, ..., n)$, which may result in the highest fitness value being assigned to a phenotype that only contains a single ELR whose similarity is the highest among all possible $ELRs$. Therefore, we consider that the aggregation function should be a balance between the *sum* and *average* of s_i $(i = 1, ..., n)$, such as $sum(s_i) \times average(s_i)$, where $i = 1, ..., n$.

However, by applying $sum(s_i)$ as a part of the aggregation function, the function tends to give a higher value to a phenotype that contains several 1-to-1 $ELRs$ rather than a phenotype that contains a single n-to-m ELR_i, even though the similarity of the n-to-m ELR_i is greater than the similarity of each of 1-to-1 $ELRs$. For example, given Phenotypes $\mathcal{P}_1$ and $\mathcal{P}_5$ in Example 5, where $\mathcal{P}_1$ is a better characterization of the relationship between RDB1 and RDB2 than $\mathcal{P}_5$:

$\mathcal{P}_1$={({home_cust, overseas_cust}, {customer, cust_phone}),
({account}, {cust_account})}

$\mathcal{P}_5$={({home_cust}, {customer}), ({overseas_cust}, {cust_phone}),
({account}, {cust_account})}

we assume the similarities of the $ELRs$ in $\mathcal{P}_1$ are 0.9 and 0.85, respectively; and the similarities of the $ELRs$ in $\mathcal{P}_5$ are 0.7, 0.7 and 0.85, respectively. Using $sum(s_i) \times average(s_i)$, the fitness value of $\mathcal{P}_1$ is $(0.9+0.85) \times (0.9+0.85)/2 = 1.53$, and the fitness value of $\mathcal{P}_5$ is $(0.7 + 0.7 + 0.85) \times (0.7 + 0.7 + 0.85)/3 = 1.685$, which will result in $\mathcal{P}_5$ being incorrectly returned. Therefore, we decided to combine the coverage (i.e., the number of entities associated by an ELR) with the $sum(s_i) \times average(s_i)$ function in the aggregation function.

Taking all factors into consideration, we derive the aggregation Function 5.3 to calculate the fitness value for the phenotype $\mathcal{P} = \{ELR_1, ELR_2, ..., ELR_n\}$:

$$\begin{aligned} sum(s_i \times c_i) \times average(s_i \times c_i) &= [\sum_{i=1}^{n}(s_i \times c_i)] \times \frac{\sum_{i=1}^{n}(s_i \times c_i)}{\sum_{i=1}^{n} c_i} \\ &= \frac{(\sum_{i=1}^{n} s_i \times c_i)^2}{\sum_{i=1}^{n} c_i} \end{aligned} \tag{5.3}$$

where s_i is the similarity of $ELR_i = (ES_i^s, ES_i^t)$, and $c_i = avg(|ES_i^s|, |ES_i^t|)$ is its coverage, in which $|ES_i^s|$ and $|ES_i^t|$ represent the number of entities in the sets ES_i^s and ES_i^t, respectively.

In the previous example, we derive the fitness value of $\mathcal{P}_1$ as $(0.9 \times 2 + 0.85 \times 1)^2/(2 + 1) = 2.34$, and the fitness value of $\mathcal{P}_5$ as $(0.7 \times 1 + 0.7 \times 1 + 0.85 \times$

$1)^2/(1+1+1) = 1.685$. Thus, $\mathcal{P}_1$ is assigned a higher fitness value.

Until now (in Sections 5.3 to 5.5), we have described the method for inferring a set of *ELRs* between source and target schemas, including the framework of searching for a particular set of *ELRs* (Section 5.3), the representations of the set of *ELRs* (Section 5.4), and the objective function (Section 5.5) that models the requirements for inferring schematic correspondences at the entity-level and assigns relative fitness values to all possible sets of *ELRs*, where the set of *ELRs* with the highest fitness value is returned, such as $\mathcal{P}_1$. Therefore, the entity-level schematic correspondences between the source and target schemas can be represented by the phenotype $\mathcal{P}_1$. However, the schematic correspondences at the attribute-level still need to be worked out, as presented in Section 5.6.

5.6 Identification of Attribute-Level Relationships

Identifying the attribute-level relationships (*ALRs*) is a post-processing step after the evolutionary search that identifies a set of *ELRs* (Sections 5.3 to 5.5), and completes the second step for inferring schematic correspondences (Algorithm 1 in Section 5.2). In this section, we identify *ALRs* for a phenotype $\mathcal{P} = \{ELR_1, ELR_2, ..., ELR_n\}$ that has the maximum fitness value among all phenotypes in the search space. In particular, we identify a set of *ALRs* between each pair of entity sets ES_i^s and ES_i^t associated by ELR_i $(i = 1, ..., n)$ to denote the equivalent 1-to-1 or n-to-m attributes. Remember that we have identified 1-to-1 matched terms while calculating the similarity of ES_i^s and ES_i^t in Section 5.5.6. The matched terms associate 1-to-1 equivalent attributes between 1-to-1 entities; they also relate 1-to-1 or n-to-m equivalent attributes between n-to-m entities. Thus, we directly utilize the 1-to-1 matched terms as *ALRs*. Example 15 in Section 5.5.6 is also an example for *ALRs*.

In this section, we mainly discuss the identification of attribute many-to-one correspondences between equivalent entity sets ES_i^s and ES_i^t associated by $ELR_i \in \mathcal{P}$. As presented in Section 1.1, an attribute many-to-one correspondence indicates that instances of a set of attributes can be translated into instances of a single attribute using a formula. We cover the common formulae enumerated by Kim *et al.* [KCGS93], which can easily be extended to other formulae, as follows:

Algorithm 9 Many2One(Attribute Set $\mathcal{A}$, Attribute Set $\mathcal{B}$, Derived Match Set $\mathcal{M}$)

```
1: $\mathbb{R} \leftarrow \emptyset$;
2: for each attribute $B \in \mathcal{B}$ do
3:     identify a set of attributes $\mathcal{A}'$ from $\mathcal{A}$, and each attribute $A_x \in \mathcal{A}'$ satisfies:
4:         (i) there exists a derived match $m \in \mathcal{M}$, where $m = \langle A_x, B, \Delta_{A_xB} \rangle$ and $\Delta_{A_xB} > 0$;
5:         (ii) attributes $A_x$ and $B$ have the same data type;
6:     $\mathcal{A}_1, ..., \mathcal{A}_n \leftarrow$all combinatorial combinations of $\mathcal{A}'$ with all lengths $\in [2, |\mathcal{A}'|]$;
7:     for each attribute set $\mathcal{A}_i = \{A_1, ..., A_k\}$, where $1 \leqslant i \leqslant n$ do
8:         for each formula $F_j$ provided do
9:             a new attribute $a_{ij} \leftarrow$ using $F_j$ on attributes in set $\mathcal{A}_i$; //transform instances of $\mathcal{A}_i$ into instances of $a_{ij}$.
10:            $\delta \leftarrow$instance similarity of attributes $a_{ij}$ and $B$;
11:            if $\delta > 0$ then
12:                $\Delta_{a_{ij}B}$ (the similarity of $\mathcal{A}_i$ and $B$ using $F_j$) $\leftarrow \delta + \Delta_{A_1B} + ... + \Delta_{A_kB}$, where $\Delta_{A_1B}, ..., \Delta_{A_kB}$ are the derived match similarities between attributes $A_1$,...,$A_k$ and $B$, respectively;
13:            else
14:                $\Delta_{a_{ij}B} \leftarrow 0.0$;
15:    $\mathbb{R} \leftarrow \mathbb{R} \cup \{(\mathcal{A}_i, B)\}$, where an attribute set $\mathcal{A}_i$ ($i \in [1, n]$) has the maximum similarity ($\Delta_{a_{ij}B} > 0$) with $B$ using a formula $F_j$ among similarities of all attribute sets $\mathcal{A}_1, ..., \mathcal{A}_n$ and $B$ using all provided formulae;
16: return $\mathbb{R}$.
```

- *string concatenation*, e.g.,
  concat (*first_name*, *last_name*) = *name*
  concat (*str_num*, *str_name*, *city*, *zip*) = *address*
- *numeric concatenation*, e.g.,
  concat (*phone_city*, *phone_area*, *phone_local*, *phone_extension*) = *phone*
- *date concatenation*, e.g., concat (*day/month/year*) = *date*

Algorithm 9 describes the process for identifying attribute many-to-one correspondences, which can also be applied to identify attribute one-to-many correspondences (omitted here). It takes as input all attributes of equivalent entities in ES_i^s and in ES_i^t (Attribute Sets $\mathcal{A}$ and $\mathcal{B}$ in Algorithm 9), and the derived matches. Algorithm 9 starts with identifying different subsets of attributes from set $\mathcal{A}$ for each attribute $B \in \mathcal{B}$, where each attribute in set $\mathcal{A}$ and attribute B are associated by a derived match whose similarity is greater than 0 and these attributes should have the same data type (Lines *2* to *6* in Algorithm 9). As we identify attribute many-to-one correspondences only between equivalent entities and restrict that attributes in each subset of $\mathcal{A}$ and attribute B are matched and have the same data type, the search of alternative subsets of $\mathcal{A}$ is restricted to a fairly small space. In particular, we look for a subset of attributes $\mathcal{A}_i$ whose instances can be transformed into the instances of attribute B using a formula (Lines *7* to *11* in Algorithm 9). Meanwhile the attribute set $\mathcal{A}_i$ should have the maximum similarity with attribute B among all attribute sets (Lines *12* to *15* in Algorithm 9). We consider such a pair of attribute set $\mathcal{A}_i$ and attribute B as an attribute many-to-one correspondence.

5.7 Summary and Conclusions

This chapter has presented an approach for inferring schematic correspondences at both entity and attribute levels. As discussed in Section 5.2, given two schemas and sets of basic matches, we employ an evolutionary search method to discover the entity-level schematic correspondences, i.e., *ELRs*, that associate pairs of equivalent entity sets, based on which we further identify specific attribute-level correspondences, i.e., *ALRs*. We have illustrated the search framework in Section 5.3, designed the representations of solutions to entity-level schematic correspondences in Section 5.4, and defined the objective function in Section 5.5. We also introduced the technique for identifying *ALRs* in Section 5.6. Specifically, we have made the following contributions:

- We have presented an approach that can identify complex many-to-many correspondences between entities and between attributes, in addition to one-to-one correspondences; whereas most existing approaches are only able to identify one-to-one correspondences, as discussed in Section 5.1.1.

- To discover entity-level schematic correspondences, we apply an evolutionary search method to allow different solutions (*ELRs*) to compete with each other, and we select the set of *ELRs* assigned the highest fitness value as the result. Thus, we do not require context specific rules, e.g., thresholds, to derive the entity-level schematic correspondences, which might have to be adjusted between applications or may require training data to identify the ideal values. By contrast, SeMap [WP08] infers complex relationships between two schemas, but uses heuristic rules to derive the results (Section

5.1.2). The method proposed by Elmeleegy *et al.* [EOE08] uses an evolutionary search method to identify 1-to-1 attribute matches (Section 5.1.3), and hence focuses on an easier problem than we do.

- We model the requirements for identifying entity-level schematic correspondences using an objective function. In particular, the requirements include: i) each single *ELR* is assigned a reasonable similarity (especially, different entities that coincidentally have common attributes should be assigned low similarities); ii) an equivalent n-to-m *ELR* has higher similarity than *ELRs* associating its subsets of entities; iii) establish partitioning of an n-to-m *ELR*; and iv) return as many *ELRs* that have comparatively high similarities as possible. To satisfy the requirements, we represent an entity or a set of entities associated by each *ELR* as a vector, and calculate the similarity of the *ELR* by comparing vectors that represent the *ELR*. We aggregate the *ELR* similarities as the fitness value of the set of *ELRs*. Methods devised by Xu *et al.* [XE06] and Melnik *et al.* [MBHR05] also identify many-to-many entity associations (Section 5.1.2), but require specific schema information (e.g., integrity constraints), which is not needed in our method.

Chapter 6

Experimental Evaluation

The research presented in this thesis aims to infer schematic correspondences that can be used for automatically generating mappings. As presented in Section 5, we have devised a method that i) infers entity-level relationships (*ELRs*) that associate 1-to-1 or n-to-m entity sets using an evolutionary search method, i.e., the genetic algorithm, and ii) identifies attribute-level relationships (*ALRs*) for each *ELR*. During the search, our method applies an aggregation function to derive an overall fitness value of *ELRs* whose similarities are calculated using an adaptation of the vector space model used in traditional information retrieval.

This chapter presents experimental studies of our method to evaluate its effectiveness. We evaluate our method using the collection of MatchBench scenarios and a set of real world relational databases provided by the Amalgam benchmark [MFH+01]. We compare results of our method only with COMA++ [DR07], because it has been demonstrated to achieve better performance than Rondo [MRB03] and OpenII [SMH+10] in Chapter 4. For the sake of simplicity, we do not repeat the results of Rondo and OpenII in this chapter.

The remainder of this chapter is structured as follows. Section 6.1 presents settings of the experimental study, including the description for the employed datasets, evaluation metrics, the genetic algorithm configuration, and basic matches we use as input of our method. Sections 6.2 and 6.3 show results of our method and COMA++ using scenarios provided by MatchBench and by the Amalgam benchmark, respectively. We conclude the chapter in Section 6.4.

6.1 Experimental Settings

6.1.1 Dataset description

As we have presented the MatchBench scenarios (the first set of scenarios) in detail in Chapter 3, we do not describe them here. The second set of scenarios we used to evaluate our method is provided by the Amalgam benchmark, which comprises four relational databases (referred to as s1, s2, s3 and s4) on the bibliographic domain devised by different designers [MFH+01]. The schemas of s1 to s4 are presented in Appendix A. We asked three experts who have good understanding of the bibliographic domain and schematic heterogeneities to manually specify the ground truth for schematic correspondences between pairwise Amalgam relational databases. As a result, we decided to use four pairs, but not others (e.g., s1/s2), of Amalgam databases as test cases, because they represent most types of schematic heterogeneities defined in Section 1.1, as presented in Table 6.1 where we list the number of ground truth occurrences of each type of schematic correspondence. Unfortunately, n-to-1 attribute correspondences are not represented in any pair of Amalgam databases.

S/T databases	the number of schematic correspondences							
	1-to-1 equivalent entities	n-to-m HP vs HP entities	n-to-m VP vs VP entities	n-to-m HP vs VP entities	n-to-m VP vs HP entities	1-to-1/n-to-m equivalent attributes	missing attributes	n-to-1 attributes
s1/s3	1	1				10	28	
s1/s4	1			1		7	64	
s2/s3	1				1	10	10	
s3/s4	1		1			6	23	

Table 6.1: The number of ground truth occurrences of each type of schematic correspondences between the Amalgam databases.

6.1.2 Metrics

To evaluate the effectiveness of our method for inferring schematic correspondences, we use the traditional metrics in schema matching, i.e., precision, recall and F-measure, as defined in Section 3.1.1.

6.1.3 Genetic algorithm setup

As stated in Section 5.3, running a genetic algorithm requires parameters for population size, offspring size, mutation rate, crossover rate and generation numbers.

In principle, we follow suggestions from text books and papers [SCED89, Gre86, ES03], did a sensitivity analysis and set population size as 30, offspring size as 30, mutation rate as $1/n$ (n represents the length of chromosome), and crossover rate as 0.9. Usually, if the search goes through more generations, it is more likely to obtain the global optimal solution. Therefore, for the small and medium scale schemas, we terminate the search when 500 generations have been produced.

The search space of all possible sets of *ELRs* is very large. For example, in MatchBench scenarios where both schemas have 6 entities, if we assume there is at least one match between each pair of entities, the search space is 2^{36} (i.e., using a binary string with the length of 6×6 as the genotype representation of a solution, as defined in Definition 4), but if both schemas have 10 entities, the search space is 2^{100}. We ran all our experiments on a 1.86 GHz Intel core2 processor with 1.0 GB RAM running Windows XP. For MatchBench scenarios, it takes 15 minutes on average to terminate the evolutionary search (500 iterations). For the Amalgam benchmark where schemas are larger than the MatchBench scenarios, it takes more than 20 minutes on average to terminate the evolutionary search (500 iterations). Due to the large search space, the proposed method is not particularly scalable.

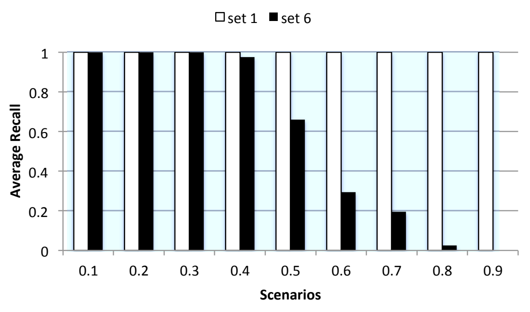
6.1.4 Basic matches

Our method for inferring schematic correspondences takes as input a source and a target schema and sets of basic matches between them that denote element similarities. As a number of schema matching approaches have been designed and implemented, we decided to utilize some of the matchers provided by COMA++ to produce the basic matches. In particular, we chose the *Name* and *Content-based* matchers, which have been demonstrated to be effective in Chapter 4, to compute the name similarities and the instance similarities of elements between the two schemas, because these two types of similarities mostly represent the commonalities and differences between the elements. We sum up similarity scores of the two sets of basic matches as evidence on which the identification of schematic correspondences can be based. Note that other types of similarity evidence, e.g., data type similarities or synonym similarities, can also be easily added up to support the identification in additional to the name and instance similarities, although they are not applied in this thesis.

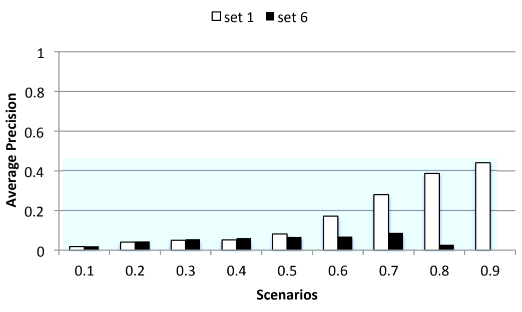
By investigating the basic matches produced by COMA++, we have noticed that almost every pair of elements between source and target schemas are matched. Some basic matches associate elements that have overlapping name strings or instance values, e.g., attributes *broker_id* and *ap_tax_id*, are matched by the *Name* matcher with similarity 0.4, which we denote as the *signal matches* since the name strings of their associated elements do overlap. However, other basic matches associate completely different elements, although with fairly low similarity scores, e.g., attributes *zip_code* and *name* are matched by the *Name* matcher with the similarity score of 0.1, which we call *noise matches*. A collection of *signal matches* plus a large collection of *noise matches* leave us a fairly large search space.

To reduce the search space and make better use of match evidence, we need to distinguish useful *signal matches* from *noise matches* by excluding basic matches that associate elements that do not exhibit commonalities. We ran a sensitivity analysis for the basic matches using a small set of scenarios from the MatchBench collection as follows (see also Figures 3.3 and 3.7 in Section 3.3). We require that equivalent entities and equivalent attributes in the chosen scenarios pertain overlapping name strings or overlapping instance values, and thus we can use matches between equivalent entities/attributes as *signal matches*, and consider others as *noise matches*. For the *Name* matcher, we choose scenarios where equivalent entities and equivalent attributes have the same names (e.g., a subset of scenarios in Set *1* of Figures 3.3) or similar names (e.g., a subset of scenarios in Set *6* of Figures 3.3); for the *Content-based* matcher, we select scenarios where 1-to-1 equivalent entities have the same instances (e.g., a subset of scenarios in Sets *1* and *3* in Figures 3.3), and n-to-m scenarios where equivalent entity sets have the same instances (e.g., a subset of scenarios in Sets *19* of Figure 3.7).

We apply the same process for excluding *noise matches* to the *Name* and *Content-based* matchers, but only show the results of the *Name* matcher as an example. In the process, we wish to obtain a threshold for the *Name* matcher to remove most *noise matches* while keeping *signal matches*. We incrementally raise the threshold on matches produced by the *Name* matcher from 0.1 to 0.9, and observe the recall and precision at each threshold, as shown in Figure 6.1. The following can be observed: (i) given the threshold of 0.4, some of true positives that have similar names are not returned (Figure 6.1(a)); (ii) precision increases from thresholds 0.1 to 0.3 (Figure 6.1(b)), while recall remains constant at 1.0



(a) Recall



(b) Precision

Figure 6.1: COMA++ *Name* matcher results given different threshold values.

(Figure 6.1(a)). As we aim to remove as many of the *noise matches* as possible while keeping as many of the *signal matches* as possible, we decided to use the threshold of 0.3 for the *Name* matcher. We did the same experiment for the *Content-based* matcher, and obtained the threshold of 0.3 as well. The threshold of 0.3 for both the *Name* and *Content-based* matchers is not a context sensitive threshold. We propose to apply such threshold not because our method requires them but because the two matchers provided by COMA++ associate attributes that have no commonalities with low similarity scores, which results in a large number of input matches, and therefore a large search space. The threshold of 0.3 for both *Name* and *Content-based* matchers will not change in different scenario contexts. The thresholds are also tested on the Amalgam scenarios, and have been demonstrated to be helpful in removing noise matches.

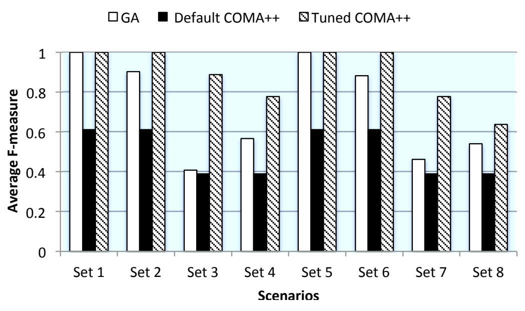
6.2 Experimental Evaluation on MatchBench

In this section, we compare the results of our method (denoted as *GA* in Figures 6.2 to 6.10) with the results of COMA++ using the following two different configuration settings. The first configuration of COMA++ is the one we used in Chapter 4, where the *threshold* and *delta* used in the *Threshold+MaxDelta* method is set as default (i.e., 0.1 and 0.01). In the second setting, we keep the majority of the parameters (e.g., matching strategy, chosen matchers and combination parameters) the same, and use as the *threshold* and *delta* the values (i.e., 0.42 and 0.33), which we had already identified as resulting in the best performance of COMA++ when applied to the MatchBench scenarios (see Chapter 4). Briefly, we call the first setting *Default* COMA++ and the second *Tuned*

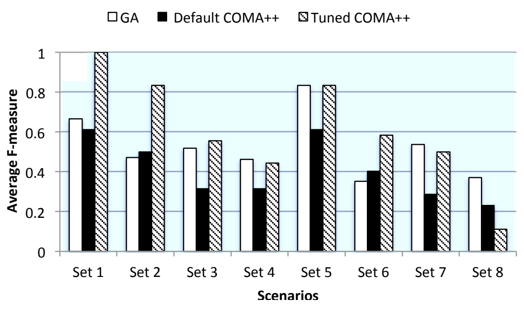
COMA++ in the remainder of this chapter. To illustrate lessons learnt in this section, we carefully investigated the experimental results and manually conducted intensive analyses in Subsection 6.2.1 to 6.2.10.

6.2.1 Experiment 1: Identifying when the same entity occurs in positive scenarios

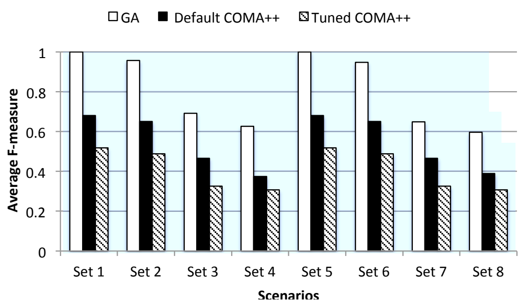
The *F-measure*, *precision* and *recall* of this experiment are presented in Figures 6.2, 6.3 and 6.4, respectively. We evaluate the overall performance of our method and COMA++ in this experiment using the *F-measure*. We also report the *precision* and *recall* for a more detailed comparison with COMA++. Compared with *Default* COMA++ and *Tuned* COMA++, our method has performed as follows:



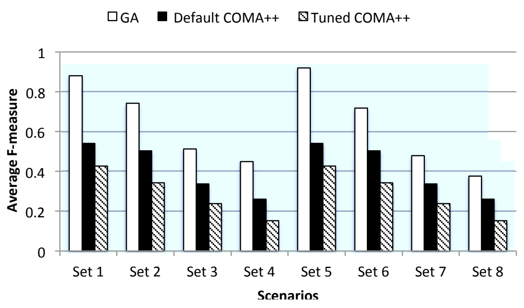
(a) Expt 1: R1, same instances



(b) Expt 1: R1, disjoint instances



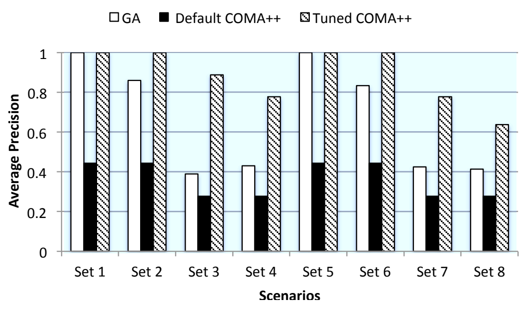
(c) Expt 1: R2, same instances



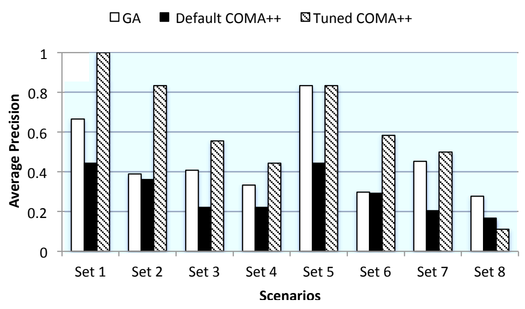
(d) Expt 1: R2, disjoint instances

Figure 6.2: Experiment 1 for Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting: F-measure.

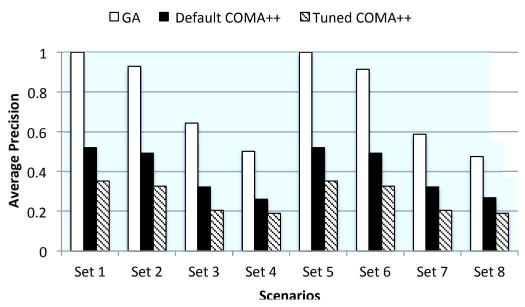
1. Our method has been generally successful at satisfying requirement *R1*, i.e., matching equivalent entities, and has performed better than *Default* COMA++, though it met this requirement less well than *Tuned* COMA++



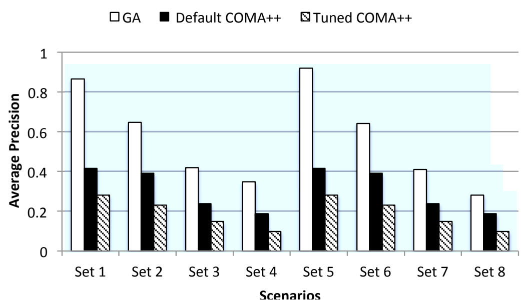
(a) Expt 1: R1, same instances



(b) Expt 1: R1, disjoint instances



(c) Expt 1: R2, same instances

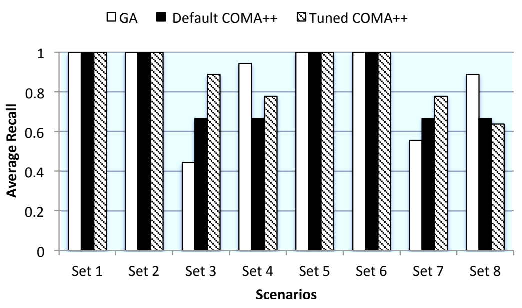


(d) Expt 1: R2, disjoint instances

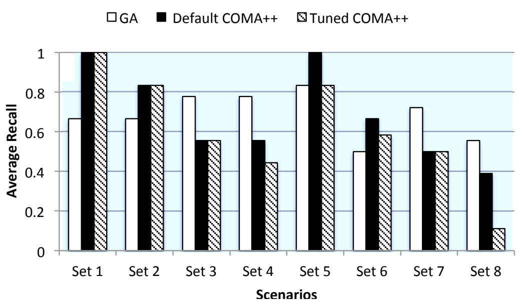
Figure 6.3: Experiment 1 for Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting: Precision.

(Figure 6.2(a) and (b)). Our method has been competitive with *Default* COMA++ in terms of *recall* (Figure 6.4(a) and (b)), but has shown significant improvement to it in terms of *precision* (Figure 6.3(a) and (b)). This is because: (i) false positives between different entities are assigned rather low similarities using the vector space model (VSM), as targeted by the objective function (Section 5.5); (ii) when the similarity between equivalent entities is high, the aggregation Function 5.3 in Section 5.5.7 is able to assign a lower fitness value to a solution that contains both false positives and the true positive than the solution that only contains the true positive, thus helping to remove false positives. *Tuned* COMA++ has outperformed our method, but relies on training data that can be used to identify a tuned threshold of 0.42, and as such can remove most false positives between different entities; by contrast, the threshold of 0.1 applied by *Default* COMA++ is rather low, which leads to several false positives being returned.

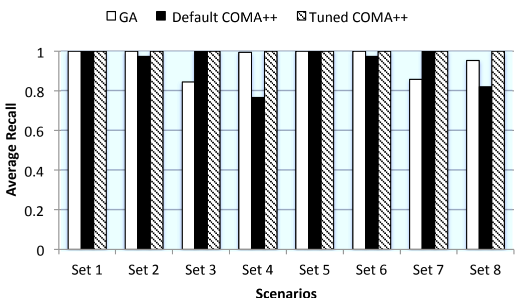
2. When only weak match evidence is available (e.g., equivalent attributes



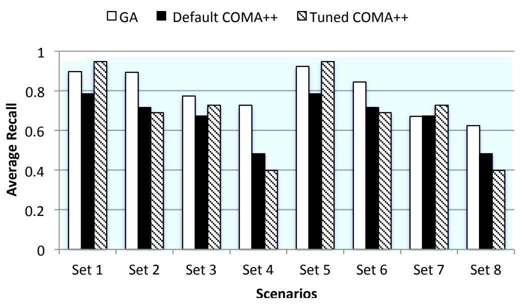
(a) Expt 1: R1, same instances



(b) Expt 1: R1, disjoint instances



(c) Expt 1: R2, same instances



(d) Expt 1: R2, disjoint instances

Figure 6.4: Experiment 1 for Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting: Recall.

have similar names and disjoint instances), our method is able to return more true positives than COMA++ (higher *recall* in Sets *4* and *8* in Figure 6.4(b) is reported for *GA* than on *Default* and *Tuned* COMA++). This is because we sum up the match evidence produced by various basic matchers (e.g., *Name* or *Instances* matchers) rather than averaging this evidence, which would dilute the match evidence. *Tuned* COMA++ performed the worst because the threshold of 0.42 resulted in several true positives that have similar names and disjoint instances being removed.

3. Our method has been much more successful than *Default* COMA++ and *Tuned* COMA++ in meeting requirement *R2*, i.e., matching equivalent attributes (Figure 6.2(c) and (d)). Although our method has been competitive with COMA++ (for both configuration settings) in associating equivalent attributes (*recall* is similar in Figure 6.4(c) and (d)), it has significantly outperformed COMA++ on removing false positives (*precision* of *GA* is much higher than *Default* COMA++ and *Tuned* COMA++ in Figure 6.3(c) and

(d)). Our method follows a top-down approach, which identifies equivalent attributes only between entities that have been identified as being equivalent, therefore, once equivalent entities are correctly associated, the chance that equivalent attributes are matched is high. When true positives of equivalent entities are not returned, it is usually because the match evidence between the equivalent entities is weak, and a different entity may have a few similar attributes with one of the equivalent entities thus being associated with them together. In such cases, equivalent attributes can also be identified successfully. This approach also helps to exclude false positives that associate one of equivalent attributes with attributes in different entities.

The reasons for the worse performance of COMA++ are as follows: (i) with a small delta of 0.01, *Default* COMA++ is able to remove false positives by associating an attribute with only a limited number of matches (e.g., a single match), but the low threshold of 0.1 is not able to remove sufficient false positives; (ii) *Tuned* COMA++ uses the delta of 0.33, and as such might associate multiple matches with a single attribute, which is not appropriate given the requirement of associating 1-to-1 attributes here. The threshold of 0.42 is able to remove some but not all false positives, and sometimes also removes true positives. This observation again indicates that in order to enable COMA to perform well, COMA++ needs to be configured appropriately for the context it is applied. This might be quite a significant challenge if schemas to be matched have different contexts throughout them. For example, some entities/attributes (e.g., scenarios used in Experiment 1) have only 1-to-1 matches whereas others (e.g., scenarios used in Experiment 9) have n-to-m matches, both of which would require a different configuration of COMA and a good knowledge of the schemas by the user who configures COMA.

Considering the *precision* and *recall* reported in Figures 6.3 and 6.4, we observe the following in terms of the performance of our method.

1. When strong evidence for matching equivalent entities is available, e.g., when they have the same instances and the same attributes, no false positives between different entities are returned (*precision* is 1.0 for Sets *1* and *5* in Figure 6.3(a)). However, when such evidence is weak, different entities might be associated (*precision* is smaller than 1.0 for Sets *2* and *6* in Figure

6.3(a), while *recall* is 1.0 in the same scenario sets in Figure 6.4(a)). This is due to the aggregation Function 5.3. Whether *ELRs* can be included to a top solution (i.e., the solution in the search space assigned the top fitness value by Function 5.3) is decided by the *ELR* that has the top similarity among all possible *ELRs*. If the top *ELR* similarity is much higher (e.g., 1.0) than similarities of other *ELRs* (e.g., 0.3), the top solution only contains the top *ELR*. By contrast, if the top *ELR* does not have much higher similarity (e.g., 0.5) than other *ELRs* (e.g., 0.3), the top solution might include other *ELRs*. In this experiment, the top *ELR* is usually the true positive that associates equivalent entities. Thus, when strong evidence for matching equivalent entities is available, the top solution would only contain the true positive without including any false positives whose *ELR* similarities are much lower than the true positive. However, when the match evidence is weak between the equivalent entities, the *ELR* that associates them has low similarity, and as such the top solution may contain both the true positive and some false positives.

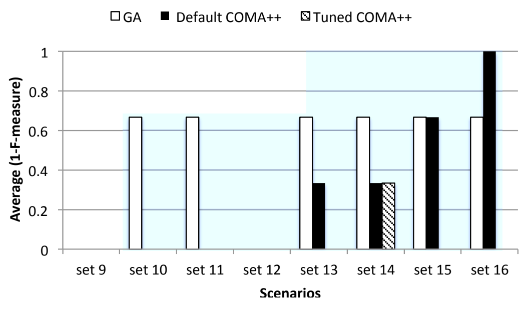
2. When match evidence for equivalent entities is weak, the top solution may contain an *ELR* that associates one of the equivalent entities with a different entity; sometimes the top solution may also have an n-to-m *ELR* that associates equivalent entities with other entities (*recall* is 1.0 only in Sets *1*, *2*, *5* and *6* in Figure 6.4(a), but is smaller than 1.0 in other scenario sets in Figure 6.4(a) and (b)). Where one of equivalent entities is matched to a different entity, it is because the match evidence between equivalent entities is rather weak, e.g., a significant number of attributes are removed and equivalent entities have disjoint instances. The case that the n-to-m entities, where the equivalent entities are a part, are associated incorrectly is because when Function 5.3 (i.e., $\frac{(\sum_{i=1}^{n} s_i \times c_i)^2}{\sum_{i=1}^{n} c_i}$) aggregates similarities (s_i) of *ELRs* in a solution, it also considers the coverages (c_i) of the *ELRs*. Thus, Function 5.3 is a tradeoff between the similarity and the coverage of each *ELR*. Assuming there are two solutions, such that the first solution contains a 1-to-1 *ELR* that associates the equivalent entities and the second solution contains an n-to-m *ELR* that associates the equivalent entities with other entities. When the similarity of the 1-to-1 *ELR* is much higher than the n-to-m *ELR*, $s_i \times c_i$ of the 1-to-1 *ELR* would be higher than the n-to-m *ELR*; however, when the similarity of the 1-to-1 *ELR* is merely a little higher than

the n-to-m *ELR*, $s_i \times c_i$ of the n-to-m *ELR* would be higher than the 1-to-1 *ELR*. Therefore, the incorrect n-to-m *ELR*, where the equivalent entities are a part, may be included in the top solution.

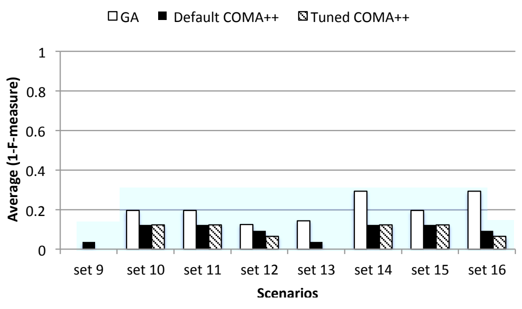
Similar to the observation in Chapter 4, we observe here that results of our method and COMA++ (for both configuration settings) between Scenario Sets *1* and *5*, *2* and *6*, *3* and *7*, and *4* and *8*, where the only difference is a change in the entity names, only differ slightly, as shown in Figure 6.2(c) and (d). Thus, we will not report the results for Scenario Sets *5* to *8* in the text describing Experiments *3* to *6* and *8* (Sections 6.2.3 to 6.2.6 and 6.2.8).

6.2.2 Experiment 2: Identifying when the same entity occurs in negative scenarios

The results of this experiment are reported in Figure 6.5. Using the same strategy as Section 4.2.2, in this experiment we only observe and measure results produced by our method and by COMA++ between the pair of different entities, where similarities (e.g., SNDE, SNSA, DNSA in Figure 3.5 in Section 3.3.3) are injected. The following can be observed:



(a) Expt 2: R1



(b) Expt 2: R2

Figure 6.5: Experiment 2 for Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting.

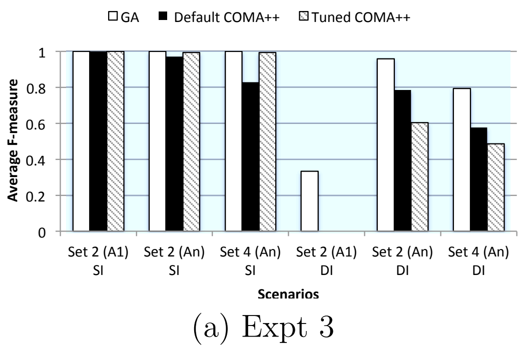
1. Our method has associated more pairs of different entities than COMA++ (*Average(1-F-measure)* reported for *GA* is higher than for *Default* COMA++ and *Tuned* COMA++ in Figure 6.5(a)), which is due to the following two reasons: (i) when the source and target schemas have little in common, the aggregation Function 5.3 assigns the top fitness value to a solution that

associates several pairs of entity sets even though their similarity scores as calculated by VSM are fairly low; (ii) our method does not utilize any threshold to remove false positives, while COMA++ with both settings employs the thresholds of 0.1 and 0.42, respectively.

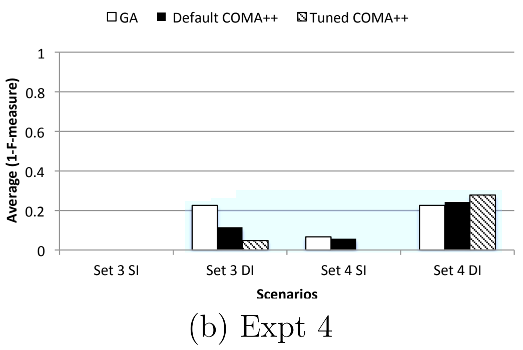
2. Our method associates more pairs of different attributes than COMA++ between the pairs of different entities into which similarities have been injected (Figure 6.5(b)). As we identify attribute-level relationships (*ALRs*) for *ELR* associated by the top solution, associations of different entities reported in Figure 6.5(a) leads to associations of different attributes.

6.2.3 Experiment 3: Identifying where different names have been given to equivalent attributes in positive scenarios

The results of this experiment are presented in Figure 6.6(a), where we distinguish *SI* and *DI* cases, and show results for *A1* and *An* cases separately, similarly to our reporting style earlier in Section 4.2.3. Our method has performed better than COMA++ in all scenario sets (*F-measure* reported in Figure 6.6(a) is higher for *GA* than for *Default* COMA++ and *Tuned* COMA++), and has been able to associate equivalent attributes that have disjoint instances, where both *Default* COMA++ and *Tuned* COMA++ have failed (Set *2 (A1) DI* in Figure 6.6(a)). Our method mostly benefits from the strategy of not applying any threshold, and as such equivalent attributes that have limited similarities can be associated.



(a) Expt 3



(b) Expt 4

Figure 6.6: Experiments 3 and 4 for Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting.

6.2.4 Experiment 4: Identifying where different names have been given to equivalent attributes in negative scenarios

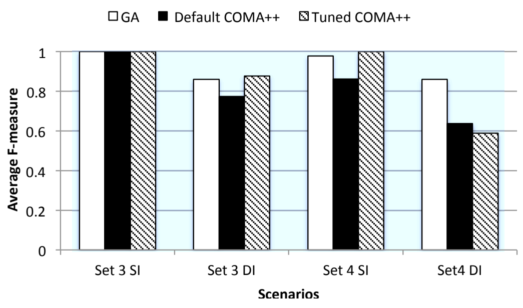
The results of this experiment are reported in Figure 6.6(b). Our method has performed slightly worse than COMA++ in that a small number of different attributes are associated incorrectly for the following reason: as mentioned in Section 6.2.1, when the match evidence is weak, e.g., equivalent attributes have similar names and equivalent entities have disjoint instances, the equivalent entities are sometimes associated incorrectly with other entities to form n-to-m associations, which can lead to different attributes being associated.

6.2.5 Experiment 5: Identifying missing attributes in positive scenarios

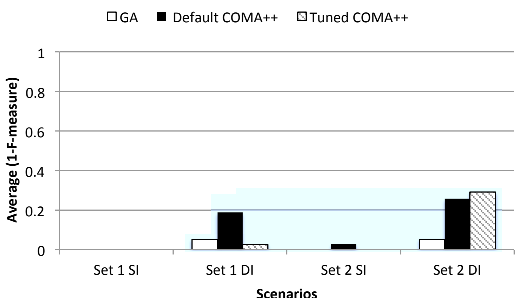
The results of this experiment are shown in Figure 6.7(a). Our method has outperformed *Default* COMA++, and is competitive with *Tuned* COMA++. Occasionally, our method has matched the missing attributes (attributes between equivalent entities that do not have corresponding attributes) to some non-missing attributes (attributes between equivalent entities that have corresponding attributes) incorrectly, when their similarity scores are higher than those of equivalent attributes (e.g., Sets *3 DI* and *4* DI in Figure 6.7 (a)). Our method has performed better than COMA++ because we do not apply a threshold to remove matches, and thus most equivalent attributes are associated. However, using the threshold COMA++ has been able to remove false positives but may also remove true positives that have low similarity scores, and thus has reported some non-missing attributes incorrectly as missing attributes.

6.2.6 Experiment 6: Identifying missing attributes in negative scenarios

The results of this experiment are presented in Figure 6.7(b). Most of non-missing attributes (attributes between equivalent entities that have corresponding attributes) are associated with an attribute by our method, whereas both *Default* COMA++ and *Tuned* COMA++ have incorrectly reported a few non-missing attributes as missing. As stated previously, our method identifies *ALRs*



(a) Expt 5



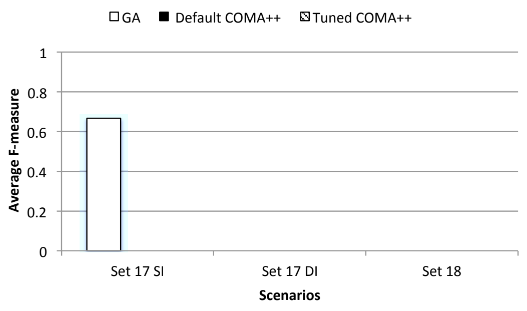
(b) Expt 6

Figure 6.7: Experiments 5 and 6 for Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting.

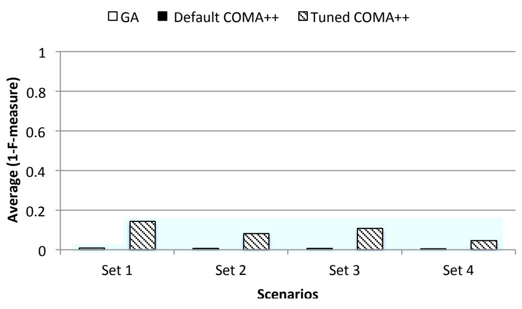
for each *ELR* associated by the top solution. Given a 1-to-1 *ELR*, *ALRs* are usually the 1-to-1 top matches between attributes. Thus, as long as the 1-to-1 equivalent entities are associated correctly, non-missing attributes would usually be associated correctly. *Default* COMA++ may associate one of the non-missing attributes with some attribute in a different entity in the other schema, if it has higher similarity with the different attribute than its corresponding attribute (*Default* COMA++ always chooses a top match among all matches between two schemas for an attribute), and thus its corresponding attribute would be incorrectly identified as a missing attribute. *Tuned* COMA++ with its threshold of 0.42 removes true positives with low similarity scores (below 0.42) (Set *2 DI* in Figure 6.7(b)), and as such equivalent attributes that are not associated after applying the threshold are incorrectly identified as missing attributes.

6.2.7 Experiment 7: Identifying many-to-one attribute correspondences in positive scenarios

The results of this experiment are shown in Figure 6.8(a). Our method has been able to identify two types of attribute many-to-one correspondences out of three types presented in Section 3.3.4, when equivalent many-to-one attributes have similar names and same instances (*F-measure* reported for Set *17 SI* in Figure 6.8(a) is the average *F-measure* of the three types). Our method requires that there is match evidence between each pair of attributes associated by the many-to-one correspondences, and that the instances of the *many* attributes can be transformed into the instances of the single attribute using the formulae presented in Section 5.6. Therefore, our method failed in Set *17 DI* where instances are disjoint between the *many* and the *one* attributes, and it failed in Set *18 SI* because no match evidence is available to support the association of equivalent many-to-one attributes (these attributes have different names and each pair of these attributes always has disjoint instances).



(a) Expt 7



(b) Expt 8

Figure 6.8: Experiments 7 and 8 for Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting.

Default COMA++ only identifies 1-to-1 matches due to its delta value of 0.1, and thus is not able to identify many-to-one associations between attributes. By contrast, *Tuned* COMA++ employs the threshold of 0.42 that has removed all true positives as they have similarity scores of less than 0.42 because the match evidence only comes from name similarities.

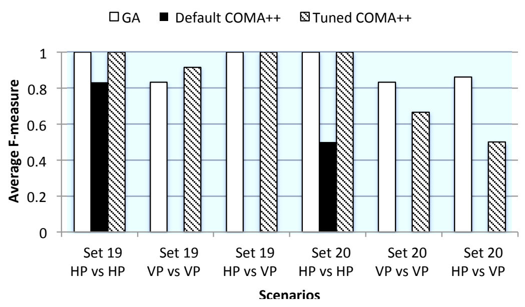
6.2.8 Experiment 8: Identifying many-to-one attribute correspondences in negative scenarios

The results of this experiment are presented in Figure 6.8(a). Our method has identified only a small number of attribute many-to-one correspondences incorrectly due to coincidental instance overlap. However, *Tuned* COMA++ has incorrectly associated several many-to-one attributes because of the delta of 0.33. Using a small delta of 0.01, *Detault* COMA++ has not associated any many-to-one attributes.

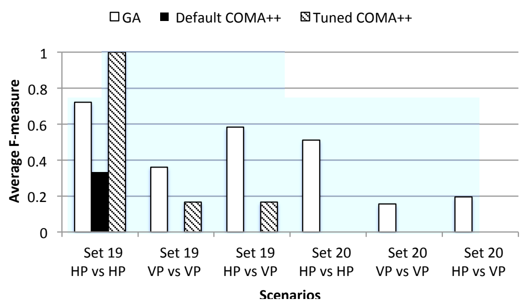
6.2.9 Experiment 9: Identifying many-to-many entity correspondences in positive scenarios

The results of this experiment are presented in Figure 6.9. Our method has outperformed *Default* COMA++ and *Tuned* COMA++ in this experiment. Our method and *Tuned* COMA++ have performed generally well when equivalent entity sets have the same instances (Figure 6.9(a), (c) and (e)), but have performed less well when there is less match evidence (due to disjoint instances) to support the identification of equivalent entity sets (Figure 6.9(b), (d) and (f)). *Default* COMA++ failed in this experiment (Figure 6.9(a) and (b)), as the default settings are more suitable to identifying 1-to-1 matches. As we have discussed results of *Default* COMA++ in detail in Section 4.2.9, we do not discuss them below. Comparing our method with *Tuned* COMA++, the following can be observed:

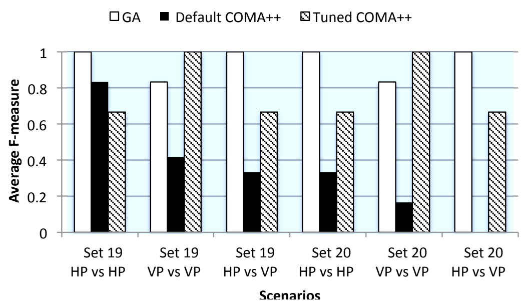
- Our method has outperformed *Tuned* COMA++ in most scenarios in terms of satisfying *Requirement R1*, namely that each of the source entity types can be matched with all the alternatively fragmented entity types. As shown in Figure 6.9(a) and (b), except for Set *19 VP vs VP* in Figure 6.9(a) and Set *20 HP vs HP* in Figure 6.9(b), higher *F-measures* are reported for our method. We discuss the reasons in the following.
    - Our method employs the vector space model (VSM) to calculate the similarity of two (sets of) entities, and enables different solutions to *ELRs* to compete with each other using the evolutionary search. The *F-measure* reported in Figure 6.9(a) demonstrates the effectiveness of our method for calculating the similarity of n-to-m entities given sufficient match evidence. Analyzing the results, the following characteristics of our method can be observed:
        1. it is rather conservative when associating equivalent n-to-m entities, and sometimes returns (n-1)-to-m entities if the number of matches between the unassociated entity and the set of *m* entities is small, or if their match similarities are low. This is most commonly the case for VP vs VP scenarios (among all partitioning types: the lowest *F-measures* are usually reported for VP vs VP in Sets *19* and *20* in Figure 6.9(a) and (b)), because the number of matches between each two VP vs VP entities is the smallest among all partitioning types. This is also the case when more differences are injected into the equivalent n-to-m entities (Set *20* in Figure 6.9(b)), resulting in low match similarities;
        2. as stated in Section 6.2.1, whether *ELRs* can be included in a top solution is decided by the *ELR* that has the top similarity among



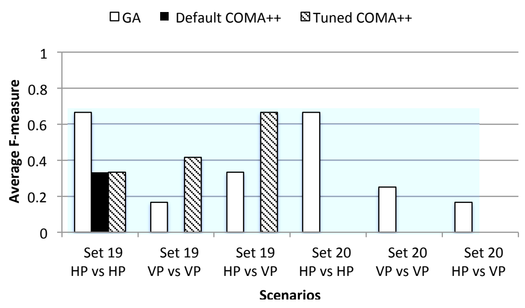
(a) R1, same instances



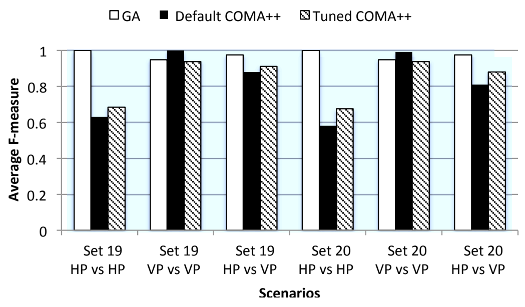
(b) R1, disjoint instances



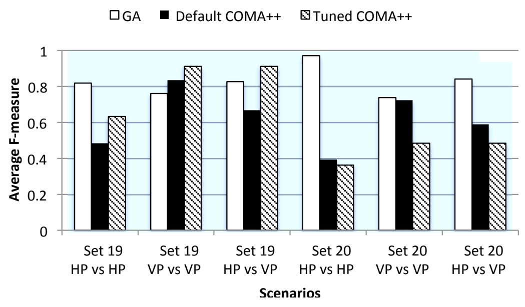
(c) R2, same instances



(d) R2, disjoint instances



(e) R3, same instances



(f) R3, disjoint instances

Figure 6.9: Experiment 9 for Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting.

all possible *ELRs*. When the similarity of the top *ELR* (usually the true positive) drops (Set *20* in Figure 6.9(a) and Set *19* in Figure 6.9(b)), the top solution may contain the top *ELR* and other *ELRs* that associate different entities (*precision* is smaller than 1.0); and (iii) additional entities may be associated incorrectly with the equivalent n-to-m entities, when VSM similarity of the equivalent n-to-m entities is fairly low, as Function 5.3 is a tradeoff between the similarity and the coverage of the associated entity sets.

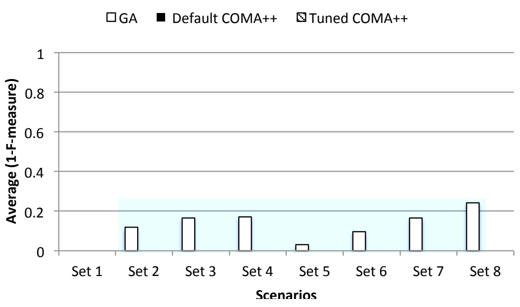
  - *Tuned* COMA++ mostly relies on the tuned delta of 0.33 that sets the similarity range for matches with a score below the score of the top match of an entity to determine matches associated with the entity, and employs the threshold of 0.42 to remove false positives. When equivalent entity sets have the same instances and their equivalent attributes have the same names (Set *19* in Figure 6.9(a)), similarities of each pair of equivalent n-to-m entities in the two schemas tend to be close and relatively high. Thus, using this tuned setting COMA++ is able to identify many-to-many associations between the n-to-m entities. However, when more differences are injected into the equivalent entity sets (Set *20* in Figure 6.9(a) and Sets *19* and *20* in Figure 6.9(b)), similarities of each pair of the equivalent n-to-m entities tend to be more different, and some of their similarities tend to be smaller than 0.42. The effects can be observed by a significant decrease of the *F-measure* reported for *Tuned* COMA++ resulting in their exclusion (Figure 6.9(a) to (b)). As the configuration setting utilized for *Tuned* COMA++ is rather MatchBench specific, *Tuned* COMA++ may fail when applied to a different set of scenarios without training data, which may not always be available.

- Our method has been competitive with *Tuned* COMA++ in satisfying *Requirement R2*, namely that primary key attributes should be matched (Figure 6.9(c) and (d)). Primary key attributes are incorrectly associated by our method when their entities have been associated incorrectly. Using the delta of 0.33, *Tuned* COMA++ has set a fairly large similarity range to return n-to-m attribute matches, and thus has associated several different attributes with the primary key attributes (Set *19* in Figure 6.9(a) and (b) and Set *20* in Figure 6.9(c)). Using the threshold of 0.42, *Tuned* COMA++ sometimes has removed almost all true positives (Set *20* in Figure 6.9(d)).

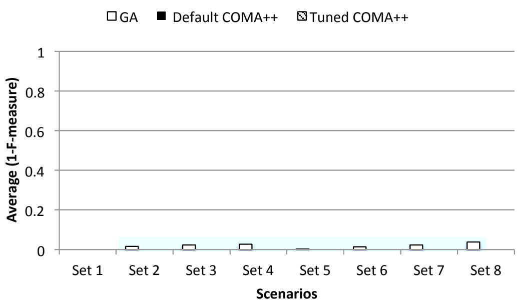
- Our method has also outperformed *Tuned* COMA++ in terms of *Requirement 3*, namely that appropriate correspondences can be identified between non-key attributes (Figure 6.9(e) and (f)). Using our method, correct n-to-m *ELRs* usually result in the correct associations of 1-to-1 or n-to-m attributes. *Tuned* COMA++ employs a context-specific setting to select matches, and the configuration setting has been determined using a training set, which means that it is successful for some scenarios (e.g., Set *19 VP vs VP* in Figure 6.9(e)), where false positives and true positives have rather different similarities, and thus false positives are removed by the delta of 0.33; however, this also results in it being less successful when applied to other scenarios that are less similar to the context used in turning thresholds (e.g., Set *20* in Figure 6.9(f)) due to the high threshold of 0.42.

6.2.10 Experiment 10: Identifying many-to-many entity correspondences in negative scenarios

The results of this experiment are presented in Figure 6.10. As can be observed, our method has identified a few incorrect n-to-m entity associations (Figure 6.10(a)), which also leads to incorrect n-to-m attribute associations (Figure 6.10(b)). This is due to the fact that the aggregation Function 5.3, as discussed in Section 6.2.1, reflects a balance between the similarity and the coverage of associated entities. When equivalent 1-to-1 entities have less in common (Sets *2* to *8* in Figure 6.10), their VSM similarity decreases. Thus, the coverage may play a more dominant role in Equation 5.3 than the similarity. The top solution may associate n-to-m entities rather than 1-to-1 entities. However, given the relatively high *F-measure* reported in Experiment 9, the performance of our method in this experiment is satisfactory. *Default* COMA++, though, has not matched incorrect n-to-m entities and attributes where it should not, but also not performed well in Experiment 9. Due to the use of MatchBench specific configuration settings, *Tuned* COMA++ has outperformed our method in this experiment. However, the training data required to identify the context specific settings may not always be available.



(a) Expt 10: R1



(b) Expt 10: R2

Figure 6.10: Experiment 10 for Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting.

6.3 Experimental Evaluation on Amalgam

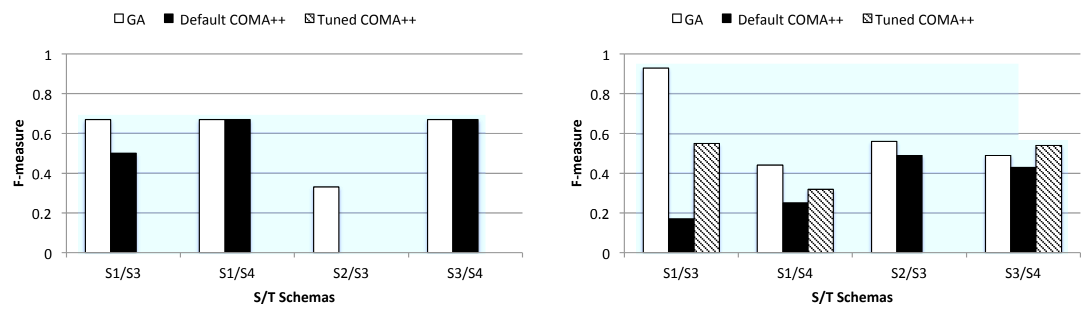
In this section, we present the experimental studies of our method using the Amalgam benchmark [MFH+01]. As introduced in Section 6.1.1, Amalgam provides four relational databases in the bibliography domain devised by independent developers, and represents various types of schematic heterogeneities, as presented in Table 6.1.

	the number of schematic correspondences						
S/T databases	1-to-1 equivalent entities	n-to-m HP vs HP entities	n-to-m VP vs VP entities	n-to-m HP vs VP entities	n-to-m VP vs HP entities	1-to-1/n-to-m equivalent attributes	missing attributes
s1/s3	1, 1, 0, **1**	1, 0, 0, **1**				10, 2, 7, **10**	28, 28, 28, **28**
s1/s4	1, 1, 0, **1**			0, 0, 0, **1**		2, 4, 3, **7**	64, 48, 60, **64**
s2/s3	0, 0, 0, **1**				1/h, 0, 0, **1**	6, 4, 0, **10**	10, 10, 10, **10**
s3/s4	1, 1, 0, **1**		0, 0, 0, **1**			2, 3, 3, **6**	23, 20, 22, **23**

Table 6.2: The number of successfully identified schematic correspondences between the Amalgam databases, following order: our method, *Default* COMA++, *Tuned* COMA++ and the ground truth.

The results of this experiment are presented in Figure 6.11 and Table 6.2. We omit the column for n-to-1 attributes in Table 6.2, as schemas provided by Amalgam do not represent this heterogeneity (see Table 6.1). We report the overall performance of our method, *Default* COMA++ and *Tuned* COMA++ using *F-measure*, i.e., *F-measure* of entity-level relationships (*ELRs*) and *F-measure* of attribute-level relationships (*ALRs*), and present the results in Figure 6.11(a) and (b), respectively. In addition, we list the number of correctly identified schematic correspondences in each cell in Table 6.2, following order: our method, *Default* COMA++, *Tuned* COMA++ and the ground truth. "1/h" in Table 6.2 means that (n-1)-to-m entities of the ground truth are associated rather than the ground truth n-to-m entities. The following can be observed:

1. Our method has outperformed *Default* COMA++ and *Tuned* COMA++ in terms of identifying *ELRs* (Figure 6.11(a)). *Default* COMA++ in turn has outperformed *Tuned* COMA++, as the latter cannot identify any of *ELRs* (the numbers of entity-level schematic correspondences identified by *Tuned* COMA++ are all 0 in Table 6.2). All the 1-to-1 *ELRs* identified by *Default* COMA++ have also been identified by our method, as reported in Column *1-to-1 equivalent entities* in Table 6.2. *Default* COMA++ cannot identify any n-to-m *ELRs*, whereas our method has associated one and a half pairs of n-to-m entities out of a total of 4 pairs. In addition, our method has incorrectly associated a few pairs of different entities (*F-measure*<1.0 is reported for our method for s1/s3 in Figure 6.11(a), even though all true positives are identified (see Table 6.2).



(a) *ELRs* (b) *ALRs*

Figure 6.11: Amalgam results by Genetic Algorithm (GA), COMA++ default setting and COMA++ tuned setting.

The databases provided by the Amalgam benchmark reflect real world scenarios. In the ground truth we have manually identified equivalent n-to-m entities, but rather little match evidence can be found between them. The first case of limited match evidence is that there are very few pairs of equivalent attributes between the equivalent n-to-m entities. For example, databases s3/s4 contain a pair of 1-to-2 VP vs VP entities, where the source and target sets have in total 16 and 15 attributes, respectively, but only 4 pairs of attributes in the two entity sets are identified as being equivalent in the ground truth (this can also be seen on the large number of missing attributes between s3/s4, as stated in Table 6.2). Therefore, our method has not been able to associate the n-to-m entities between s3/s4 (Table 6.2). For the same reason our method has failed to identify the equivalent n-to-m entities between s1/s4 and was only able to identify half of the equivalent n-to-m entities between s2/s3. The second case of limited availability of sufficient match evidence is that equivalent attributes have fairly little information in common. In the real world scenarios representing the Amalgam benchmark, equivalent entities have little overlap at the instance level, and thus the match evidence that can be used by our method is the name similarity. Where equivalent attributes have similar names (e.g., the n-to-m HP vs HP entities between s1/s3), our method has performed fairly well where the pair of n-to-m entities is correctly identified; however, when the names of equivalent attributes have little in common, e.g., the attribute that represents *abstract* in bibliographic domain is called *txt* in database s2 but is called *abstract* in database s3, our method has failed.

Default COMA++ has performed well at identifying 1-to-1 entities, as discussed above. For application on the Amalgam benchmark, the fairly low threshold of 0.1 used by *Default* COMA++ appears to be suitable for retaining the true positives between entities, even though their similarities are quite low due to the disjoint instances between most pairs of equivalent attributes. *Tuned* COMA++ configured with the MatchBench-specific configuration setting in form of threshold of 0.42 is unsuitably high for the Amalgam benchmark and, therefore, results in the removal of all true positives between entities. This indicates that: (i) the default setting suggested by COMA++ developers appears to be suitable for the application on the Amalgam benchmark and in contexts with similar properties, i.e., match evidence of limited strength; (ii) to ensure the best performance of COMA++, training data is required to determine the most appropriate configuration setting for the specific context in which it is to be applied.

2. Our method has generally been more successful than COMA++ in associating *ALRs* (Figure 6.11(b)). When equivalent entities, irrespective of 1-to-1 or n-to-m, have been associated by our method (e.g., s1/s3), it has usually been able to associate equivalent attributes as well (*F-measure* of our method reported for s1/s3 is quite good in Figure 6.11(b)). As stated previously, the effectiveness of our method on matching equivalent attributes mostly depends on the *ELRs* results. *Tuned* COMA++ has outperformed

Default COMA++ in associating *ALRs* between all pairs of databases, except s2/s3, because the delta of 0.33 used by *Tuned* COMA++ allows more attributes to be associated with an attribute, thus resulting in the association of n-to-m equivalent attributes rather than just 1-to-1 attributes. *Default* COMA++, on the other hand, has only returned 1-to-1 attributes due to the small delta of 0.1, which only tends to retain the top match for an attribute.

6.4 Summary and Conclusions

In this chapter, we evaluated our method for inferring schematic correspondences using a collection of synthetic scenarios generated using MatchBench and a collection of real world relational databases provided by the Amalgam benchmark, both of which feature various representative types of schematic correspondences between source and target schemas. We compared the results of our method with the results of COMA++ configured with the default settings, i.e., (threshold, delta) = (0.1, 0.01), and configured with the settings that we tuned specifically using MatchBench scenarios, i.e., (threshold, delta) = (0.42, 0.33).

In general, our method outperformed *Default* COMA++ when applied to both MatchBench and Amalgam scenarios in terms of its ability to inferring schematic correspondences. Our method has been competitive with *Tuned* COMA++ (with its MatchBench-specific setting) in the experimental studies on MatchBench scenarios. However, our method has been more successful than *Tuned* COMA++ on the Amalgam scenarios.

Compared to COMA++, which needs to rely on a context-specific setting to be effective in a given matching task, our method is not sensitive to its settings and has shown consistent effectiveness in inferring schematic correspondences in different domains, e.g., MatchBench and Amalgam. As presented in Sections 6.2 and 6.3, when the input matches are able to provide enough similarity evidence to support the search, our method has always been successful in identifying schematic correspondences. However, our method tends to be less effective in inferring schematic correspondences where the quality of matches is poor, such as s1/s4 schemas in the Amalgam benchmark. In summary, our method has shown the following features:

- Our method does not require a context-specific threshold to be specified by the user to retain true positives or to remove false positives. Instead, our method employs an objective function to rank different solutions and returns the solution with the top fitness value specifying the entity-level relationships (*ELRs*) identified between source and target schemas.

- Our method takes as input the source and target schemas and in addition different sets of matches (e.g., a set of name matches and a set of instance matches). Therefore, the quality of the input matches is important to our method. We anticipate that with more accurate matches (e.g., matches that associate attributes whose names are synonyms) our method would perform better on the real world scenarios provided by the Amalgam benchmark.
- Our method sums up the similarity scores of different sets of matches and does not use a threshold to select matches, and as such has been able to associate attributes whose similarity is weak (e.g., similar names and disjoint instances); by contrast, as COMA++ averages the similarity scores and uses a threshold, matches with lower similarity scores are removed prematurely.
- The objective function utilizes an adapted vector space model (VSM) to calculate the similarity of two (sets of) entities, and is able to assign a reasonable VSM similarity to a pair of entity sets using the input match evidence. VSM similarities of most pairs of equivalent entities can be distinguished from VSM similarities of most pairs of different entities, thus helping to effectively keep true positives and remove false positives using the aggregation Function 5.3 in Section 5.5.7.
- The objective function allows both 1-to-1 and n-to-m entities to be associated in the top solution.
- Following a top down method for identifying attribute-level relationships (*ALRs*), i.e., *ALRs* are identified for each *ELR* associated by the top solution, our method has achieved fairly good performance in matching equivalent 1-to-1 and n-to-m attributes.

Chapter 7

Conclusion

In this chapter, we summarize the major research results and contributions presented in this thesis in Section 7.1, and discuss the open issues we have encountered while conducting the current research as the future work in Section 7.2.

7.1 Significance of Results and Contributions

Dataspaces aim to reduce the high upfront effort required to set up a traditional data integration system. Thus, a dataspace management system is defined as a data integration system that can be set up automatically and can be incrementally improved by user's annotations. However, most existing methods for specifying views require manual effort, and thus are not suitable for setting up a dataspace management system. This is because the majority of existing schema matching approaches, such as [DR07, MGMR02, BN05, MBR01], match *one-to-one* elements that have similar names, instances, data types, or structures between source and target schemas. A well-known issue with these approaches is that the meaning carried by these matches is unspecified and ambiguous [BM07, MHH00], and therefore leads to the situation in which most methods for generating views rely heavily on external resources (e.g., rich specfication for the relationships of schemas) or human interaction [FHH$^+$09, MBHR05, PB08, XE06].

Thus, the research presented in this thesis was motivated by the idea of inferring schematic correspondences between heterogeneous but interrelated data sources, which in turn are used as input of a method for automatically specifying views [MBPF09] and bootstrap the setup process of a dataspace management system. Specifically, schematic correspondences characterize *one-to-one* or *many-to-many* equivalent relationships between elements (e.g., tables and attributes in relational databases) of source and target schemas, as discussed in Section 1.1.

Our research aimed to identify schematic correspondences that are expressive enough to serve as the basis for the automatic generation of views. By surveying the state-of-the-art on schema matching techniques, we have established that existing approaches have been good at identifying and combining similarity evidence (e.g., name and instances) between *one-to-one* elements, as presented in Chapter 2. However, their ability to identify correspondences that carry more information than the simple similarity evidence, such as schematic correspondences, has not been evaluated previously. We aimed to identify whether or not the existing schema matching approaches are able to infer schematic correspondences, and thus helped to identify whether or not the output of the existing schema matching approaches can be used as the input of the method for automatically specifying views [MBPF09], as required by dataspaces. We built our research upon the existing schema matching approaches, as our focus was not to propose a more solid method for collecting various kinds of similarity evidence, but to devise a method for effectively making use of such similarity evidence, in order to infer more expressive information that characterizes relationships between the schemas and that therefore can be used to generate views in dataspaces. Thus, we have conducted two-step research to achieve our overall aim:

1. We have evaluated the effectiveness of existing schema matching approaches on identifying schematic correspondences and developed a benchmark for this purpose, namely MatchBench. By doing this, we gained an understanding of the advantages and disadvantages of underlying techniques of the existing approaches, and thus were able to identify a collection of matchers that can be employed for inferring schematic correspondences. We have conducted a series of experiments using three well-known and publicly available schema matching platforms, and have summarized their performance into lessons on necessary steps to overcome the limitations of the existing schema matching techniques on inferring schematic correspondences.

2. Based on the lessons we have learnt from the conducted experiments, we have devised a method for inferring schematic correspondences using evolutionary search, i.e., a genetic algorithm, which allows different solutions to

compete with each other, and does not utilize heuristic rules, e.g., thresholds, to identify a solution to schematic correspondences in light of the available evidence. We have designed a novel objective function for the search method based on the vector space model. We have demonstrated the effectiveness of our method using a set of experiments.

In summary, the major contributions of the research presented in this thesis are:

A wide range of scenarios that manifest schematic heterogeneities
We have developed a collection of scenarios in MatchBench based on schematic heterogeneities summarized by Kim *et al.* [KS91], as presented in Section 3.3. MatchBench starts with two relational databases that contain a pair of exactly equivalent entities (i.e., tables), and systematically injects perturbations (e.g., various types of schematic heterogeneities presented in Section 1.1) into the pair of equivalent entities. MatchBench offers three scenario spaces where equivalent entities exhibit various types of schematic heterogeneities: a space of scenarios where *one-to-one* equivalent entities either represent each of *DNSE*, *DNSA* and *missing attributes* conflicts or represent a combination of these conflicts; a space of scenarios where *one-to-one* equivalent entities represent various types of *attribute many-to-one* conflict; and a space of scenarios where *many-to-many* equivalent entities represent *HP vs HP*, *HP vs VP* and *VP vs VP* conflicts combined with changes of attribute names. In addition, MatchBench also provides a space for different entities. It starts with two relational databases where the pair of equivalent entities are removed, and systematically injects similarities (e.g., *SNDE*, *SNDA* and *SNSA*) into a pair of different entities.

A collection of experiments over the scenarios
We have designed experiments to evaluate the effectiveness of schema matching platforms in identifying *one-to-one* equivalent entities (referring to as *DNSE*), *DNSA*, *missing attributes*, *many-to-one* equivalent attributes and *many-to-many* equivalent entities (containing *HP vs HP*, *HP vs VP* and *VP vs VP* conflicts), as presented in Section 3.4. For each above type of schematic correspondences, we have designed a positive and a negative experiment. The positive experiment evaluates matching platforms on scenarios where the schematic correspondence is present, while the negative experiment selects scenarios where the correspondence

is absent, thus requiring that the platforms should not identify such correspondences.

An evaluation of schema matching systems using MatchBench
We have applied MatchBench to evaluate three well-known and publicly available schema matching platforms, namely COMA++ [DR07], Rondo [MRB03], and OpenII [SMH$^+$10], as presented in Chapter 4. COMA++ has performed fairly well on comparing elements and determining their similarity. However, as a result of having to use heuristic choices for its configuration, COMA++'s results suffer from being sensitive to specific configurations. To account for this fact in the MatchBench experiments, we followed the instruction of COMA++ authors, and used its default setting (i.e., threshold=0.1, delta =0.01) to identify matches on MatchBench scenarios, as discussed in Section 4.1.1. This setting is mostly intended for identifying *one-to-one* matches and has failed in most of experiments where *many-to-many* matches are expected. Rondo is a model management platform providing operations that act on schemas and the correspondences between them, such as *Merge*, *Compose* and *Difference*. It employs *Similarity Flooding* [MGMR02] to identify *one-to-one* matches between two schemas, and therefore is less suitable for identifying *many-to-many* schematic correspondences. OpenII specializes in providing an interactive user interface for data integration tasks, such as schema matching. As suggested by OpenII developers, we chose to use the best match associated with each element as OpenII results, which leads to a significant number of false positives being returned. In summary, the method of selecting candidate matches directly affects the quality of the final matching results. However, although the existing schema matching approaches have performed fairly well in comparing elements and producing candidate matches, none of the three evaluated platforms offers an effective strategy for selecting candidate matches.

An evolutionary search method for inferring schematic correspondences
Given basic matches that associate elements that have similar names or instances, we have developed a two-step method for inferring schematic correspondences between source and target schemas, as described in Chapter 5: i) using an evolutionary search to infer a set of entity-level relationships (*ELRs*) between two relational databases, and ii) deriving a set of attribute-level relationships (*ALRs*)

for the *ELRs* identified in the first step. By applying the evolutionary search, we do not need to apply heuristic rules to select equivalent entities from candidate matches. Specifically, we model our requirement of identifying schematic correspondences as an objective function, which calculates the relative fitness value of a solution in the search space, thus allowing different solutions (*ELRs*) to compete with each other. The solution that is obtained during the search and has a greater fitness value than any of the other solutions is obtained as the result *ELRs*. To apply the evolutionary search, we have implemented various operators required for the evolutionary search, e.g., mutation, crossover and selection, as presented in Section 5.3, and have designed the phenotype and genotype representations for a set of *ELRs* in Section 5.4.

A novel objective function

To guide the search process, we model the requirements for inferring a set of *ELRs* using an objective function in Section 5.5. Our requirements include: i) assigning a similarity score to a pair of entity sets sufficient to differentiate between a pair of equivalent entity sets and a pair of different entity sets that coincidentally have overlapping attribute names or instances; ii) assigning a higher similarity to equivalent n-to-m entities than its subsets of entities, e.g., (n-1)-to-m entities; iii) determining partitioning of a pair of n-to-m entity sets, i.e., horizontal partitioning (*HP*) or vertical partitioning (*VP*); and iv) selecting as many pairwise entity sets that have high similarities as possible. To meet requirements i) and ii), we follow the intuition behind the design of the vector space model to calculate the similarity of pairwise entity sets. In particular, we represent two entity sets associated by an *ELR* as two vectors (Section 5.5.4) and calculate the vector similarity as their similarity (Section 5.5.6). To meet requirement iii), we express each of the two entity sets (if cardinality > 1) as being horizontal and vertical vectors, respectively, as presented in Secton 5.5.4. Choosing the maximum similarity among similarities of *HP vs HP*, *HP vs VP*, *VP vs HP*, and *VP vs VP* vectors allows us to discover the specific partitioning of the two entity sets. To meet requirement iv), we devise an aggregation function to calculate an overall similarity for a set of *ELRs* given their similarities, as presented in Section 5.5.7. This function assigns a higher overall similarity to a set of *ELRs* if it contains more *ELRs* with similarities closer to the highest *ELR* similarity. The set of *ELRs* that has the maximum aggregation similarity obtained during the search is considered as the result *ELRs*.

An experimental explanation of the effectiveness of our approach for inferring schematic correspondences
We evaluated our approach using the synthetic scenarios generated by MatchBench, and four Amalgam [MFH$^+$01] relational databases devised by independent designers. All these test cases contain various schematic correspondences between source and target schemas. We compare the performance of our approach with COMA++ configured with two different settings, which differ in the values chosen for the two parameters *Threshold* and *Delta*. The first setting, so-called *Default COMA++* in Chapter 6, uses *Threshold*=0.1 and *Delta*=0.01, is the default setting of COMA++ tool we obtained from COMA++ developers; the second setting, so-called *Tuned COMA++* in Chapter 6, uses *Threshold*=0.42 and *Delta*=0.33 which we identified as the best settings for applying COMA++ on MatchBench scenarios. Our work has been demonstrated to be effective for inferring schematic correspondences. It has outperformed *Default COMA++* for both MatchBench and Amalgam scenarios, and has been able to identify most of the schematic correspondences given sufficient match evidence (e.g., the same instances between equivalent entities). It has performed slightly worse than *Tuned COMA++* as it sometimes returns more false positives than *Tuned COMA++*, while *Tuned COMA++* that utilizes the MatchBench-specific setting is able to return true positives and remove most false positives more effectively than our approach, which has not been tuned specifically for the MatchBench scenarios. However, our method has performed much better than *Tuned COMA++*, which was tuned specifically for the MatchBench scenarios, in identifying schematic correspondences between Amalgam databases. Our method meets the scalability issue and should be improved in future work, as stated in the next section.

7.2 Future Work

In this thesis we have not investigated the complete list of schematic heterogeneities proposed by Kim *et al.* [KS91, KCGS93]. Furthermore, there are alternative ways of characterizing relationships between elements in two schemas, in addition to the schematic correspondences. These remarks, therefore, point to a number of open issues that could be explored in future work as follows:

- Covering a wider range of schematic heterogeneities. In this thesis, we have focussed our research on various types of schematic heterogeneities specified between two interrelated but heterogeneous relational databases at the schema-level. There are several types of instance-level conflicts [KS91] not addressed here, such as different instance representations for similar attributes (e.g., 06.2011 vs June 2011). Furthermore, data model heterogeneities and schematic heterogeneities tend to occur together in real world scenarios, as stated in the beginning of this thesis (Section 1.1), and thus may require further attention. Additional types of heterogeneities between relational databases and object-oriented databases have been classified by Kim *et al.* [KCGS93]. B. S. Lerner also categorizes various heterogeneities due to modifications to databases when their schemas evolve over time [Ler00]. These classifications may be extended to capture more types of semantic relationships between two interrelated but heterogeneous data sources, and may also be utilized as guidance for the inference of associations between the data sources.

- Developing and implementing a more general benchmark for diagnosing various types of heterogeneities, e.g., combined schematic heterogeneities and data model heterogeneities. A new benchmark could eliminate the limitations of the current MatchBench, which only generates scenarios between pairwise relational schemas. An improved MatchBench could take as input a single data source represented using any model, not just relational, and automatically generate scenarios that represent schematic heterogeneities and data model heterogeneities between data sources.

- An inference method that is able to infer schematic correspondences between pairwise data sources represented using different data models (e.g., relational, XSD, object, ontology, etc.). A possible approach is to utilize a model-generic method, e.g., using a canonical model, to capture common features of different data models and infer schematic correspondences between the two canonical models [HBM$^+$]. This method currently supports a generalization of relational, XSD, object and object-relational schemas. However, data sources may be represented by ontologies, and schematic correspondences may be present between a traditional database and an ontology or between two ontologies. To the best of our knowledge, this research problem has not been addressed yet.

- An improvement to the quality of input matches. Several existing schema matching approaches have been developed to identify various similarities (e.g., names, instances and data types) between elements of source and target schemas, so-called matches, as presented in Chapter 2. Due to the large corpus of previous work on schema matching, the method for inferring schematic correspondences we devised in Chapter 5 mostly contributes to how to utilize these matches to derive more information (i.e., schematic correspondence) rather than competing with previous work (i.e., identifying matches). However, the quality of input matches does affect the overall performance of the inference work. In the real world applications, e.g., those provided by the Amalgam scenarios used in Section 6.3, it appears to be rather common that attributes representing the same real world concept have little overlap on their names and instances, and thus tend not to be matched. In those cases without sufficient match evidence, our approach for inferring schematic correspondences from matches does not perform well. Therefore, we suggest that additional efforts to recognize similar attributes whose names represent the same meaning but are denoted by different strings or whose instances belong to the same domain but are disjoint are necessary to improve the performance of our approach. It is also possible to apply the pay-as-you-go approach [FHM05] that utilizes user feedback to refine the input matches, and thus help to achieve more accurate results of inferring schematic correspondences.

- Identification of more types of many-to-many entity associations in addition to horizontal partitioning and vertical partitioning. Following the classification of Kim *et al.* [KS91], many-to-many entities that represent horizontal or vertical partitioning are associated by our approach so as to show that the two sets of entities represent the same real world concept. In fact, two sets of entities that do not satisfy the requirements of horizontal or vertical partitioning may also represent the same real world information, and thus should be identified as being equivalent. For example, entities seldom appear alone and are usually associated with relationships in entity-relationship (ER) models. In such cases, more general many-to-many entity-relationship associations may capture the equivalence relationships between source and target ER models more precisely than horizontal or vertical partitioning. Thus, more work is required for summarizing and inferring these more general relationships.

- Scalability of the evolutionary search. The approach presented in Chapter 5 using an evolutionary search method is mostly suitable for inferring schematic correspondences between small scale relational databases. This is because as the number of entities in a schema increases, the search space increases exponentially. The large search space usually requires more generations/iterations to identify an optimal solution and may cause the search to get trapped in a local optimal solution rather than returning a global optimal solution. Thus, one way to infer schematic correspondences between large scale schemas is to partition the schemas into segments, and employ the evolutionary search to infer the correspondences between similar segments. Techniques used by, e.g., COMA++ [DR07] and Falcon [HQC08], may be investigated to address this issue.

- Similar to schema matching approaches, the method for inferring schematic correspondences may also be suitable to address issues of interdisciplinary research, such as bioinformatics and medicine, where data integration techniques are highly demanded [GS08, LMMS$^+$07, KN04]. Of course, further research combining both domain knowledge and our approach would be necessary.

Bibliography

[ABBG09] Paolo Atzeni, Luigi Bellomarini, Francesca Bugiotti, and Giorgio Gianforme. A runtime approach to model-independent schema and data translation. In *EDBT*, pages 275–286, 2009.

[ABMM07] Yuan An, Alexander Borgida, Renée J. Miller, and John Mylopoulos. A semantic approach to discovering schema mapping expressions. In *ICDE*, pages 206–215, 2007.

[ACPS96] Sibel Adali, K. Selçuk Candan, Yannis Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *SIGMOD Conference*, pages 137–148, 1996.

[ALM09] Shun'ichi Amano, Leonid Libkin, and Filip Murlak. Xml schema mappings. In *PODS*, pages 33–42, 2009.

[ASS09] Alsayed Algergawy, Eike Schallehn, and Gunter Saake. Improving xml schema matching performance using prüfer sequences. *Data Knowl. Eng.*, 68(8):728–747, 2009.

[ATV08] Bogdan Alexe, Wang Chiew Tan, and Yannis Velegrakis. Stbenchmark: towards a benchmark for mapping systems. *PVLDB*, 1(1):230–244, 2008.

[BBC$^+$99] Philip A. Bernstein, Thomas Bergstraesser, Jason Carlson, Shankar Pal, Paul Sanders, and David Shutt. Microsoft repository version 2 and the open information model. *Inf. Syst.*, 24(2):71–98, 1999.

[BBR11] Zohra Bellahsene, Angela Bonifati, and Erhard (Eds.) Rahm. *Towards large-scale schema and ontology matching*, pages 3–28. Schema Matching and Mapping. Springer, 2011.

[BDG$^+$07] Lukas Blunschi, Jens-Peter Dittrich, Olivier René Girard, Shant Kirakos Karakashian, and Marcos Antonio Vaz Salles. A dataspace odyssey: The imemex personal dataspace management system (demo). *CIDR*, pages 114–119, 2007.

[BEFF06] Philip Bohannon, Eiman Elnahrawy, Wenfei Fan, and Michael Flaster. Putting context into schema matching. *VLDB*, pages 307–318, 2006.

[BEG$^+$06] Bishwaranjan Bhattacharjee, Vuk Ercegovac, Joseph S. Glider, Richard A. Golding, Guy M. Lohman, Volker Markl, Hamid Pirahesh, Jun Rao, Robert M. Rees, Frederick Reiss, Eugene J. Shekita, and Garret Swart. Impliance: A next generation information management appliance. *CoRR*, abs/cs/0612129, 2006.

[Ber03] Philip A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

[BHP00] Philip A. Bernstein, Alon Y. Halevy, and Rachel Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[BLR97] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *PODS*, pages 99–108, 1997.

[BM07] Philip A. Bernstein and Sergey Melnik. Model management 2.0: manipulating richer mappings. pages 1–12, 2007.

[BN05] Alexander Bilke and Felix Naumann. Schema matching using duplicates. *ICDE*, pages 69–80, 2005.

[BPE$^+$10] Khalid Belhajjame, Norman W. Paton, Suzanne M. Embury, Alvaro A. A. Fernandes, and Cornelia Hedeler. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In *EDBT*, pages 573–584, 2010.

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[BV84] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern information retrieval.* ACM press New York, 1999.

[CFM06] Silvana Castano, Alfio Ferrara, and Stefano Montanelli. Matching ontologies in open networked systems: Techniques and applications. pages 25–63, 2006.

[CHT05] Kajal T. Claypool, Vaishali Hegde, and Naiyana Tansalarak. QMatch - a hybrid match algorithm for xml schemas. In *ICDE Workshops*, page 1281, 2005.

[CHW$^+$08] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu 0002, and Yang Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[CKP08] Laura Chiticariu, Phokion G. Kolaitis, and Lucian Popa. Interactive generation of integrated schemas. In *SIGMOD Conference*, pages 833–846, 2008.

[CRF03] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[DBH07] Fabien Duchateau, Zohra Bellahsene, and Ela Hunt. XBenchmatch: a benchmark for XML schema matching tools. *VLDB*, pages 1318–1321, 2007.

[DDH01] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. pages 509–520, 2001.

[DGL00] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive query plans for data integration. *J. Log. Program.*, 43(1):49–73, 2000.

[DHY07] Xin Luna Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. *VLDB*, pages 687–698, 2007.

[DKS$^+$08] Bing Tian Dai, Nick Koudas, Divesh Srivastava, Anthony K. H. Tung, and Suresh Venkatasubramanian. Validating multi-column schema matchings by type. *ICDE*, pages 120–129, 2008.

[DLD$^+$04] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Y. Halevy, and Pedro Domingos. imap: Discovering complex mappings between database schemas. In *SIGMOD Conference*, pages 383–394, 2004.

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Y. Halevy. Learning to map between ontologies on the semantic web. pages 662–673, 2002.

[DMR02] Hong Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. *Web, Web-Services, and Database Systems*, pages 221–237, 2002.

[Do06] Hong Hai Do. Schema Matching and Mapping-based Data Integration. *Verlag Dr. Müller (VDM)*, 2006.

[DR02] Hong Hai Do and Erhard Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. *VLDB*, pages 610–621, 2002.

[DR07] Hong Hai Do and Erhard Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007.

[DS06] Jens-Peter Dittrich and Marcos Antonio Vaz Salles. iDM: A Unified and Versatile Data Model for Personal Dataspace Management. In *VLDB*, pages 367–378, 2006.

[EFH$^+$09] Jérôme Euzenat, Alfio Ferrara, Laura Hollink, Antoine Isaac, Cliff Joslyn, Véronique Malaisé, Christian Meilicke, Andriy Nikolov, Juan Pane, Marta Sabou, François Scharffe, Pavel Shvaiko, Vassilis Spiliopoulos, Heiner Stuckenschmidt, Ondrej Sváb-Zamazal, Vojtech Svátek, Cássia Trojahn dos Santos, George A. Vouros, and Shenghui Wang. Results of the ontology alignment evaluation initiative 2009. In *OM*, 2009.

[EM07] Daniel Engmann and Sabine Maßmann. Instance Matching with COMA++. *BTW Workshops*, pages 28–37, 2007.

[Emb97] David W. Embley. *Object database Development: concepts and principles.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1997.

[EMS+06] Jérôme Euzenat, Malgorzata Mochol, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej Sváb, Vojtech Svátek, Willem Robert van Hage, and Mikalai Yatskevich. Results of the ontology alignment evaluation initiative 2006. In *Ontology Matching*, 2006.

[EOE08] Hazem Elmeleegy, Mourad Ouzzani, and Ahmed K. Elmagarmid. Usage-based schema matching. In *ICDE*, pages 20–29, 2008.

[ES03] Agoston E. Eiben and James E. Smith. *Introduction to evolutionary computing.* Springer Verlag, 2003.

[FHH+09] Ronald Fagin, Laura M. Haas, Mauricio A. Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications*, pages 198–236, 2009.

[FHM05] Michael J. Franklin, Alon Y. Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *ACM SIGMOD Record*, 34(4):27–33, 2005.

[FKMP03] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *ICDT*, pages 207–224, 2003.

[FKPT04] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. In *PODS*, pages 83–94, 2004.

[FLM99] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational plans for data integration. In *AAAI/IAAI*, pages 67–73, 1999.

[FW97] Marc Friedman and Daniel S. Weld. Efficiently executing information-gathering plans. In *IJCAI (1)*, pages 785–791, 1997.

[GMPQ$^+$97] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The tsimmis approach to mediation: Data models and languages. *J. Intell. Inf. Syst.*, 8(2):117–132, 1997.

[Gol89] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-wesley Reading Menlo Park, 1989.

[Gre86] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122–128, 1986.

[GS08] Carole A. Goble and Robert Stevens. State of the nation in data integration for bioinformatics. *Journal of Biomedical Informatics*, 41(5):687–693, 2008.

[GSY04] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *ESWS*, pages 61–75, 2004.

[GYS07] Fausto Giunchiglia, Mikalai Yatskevich, and Pavel Shvaiko. Semantic matching: Algorithms and implementation. *J. Data Semantics*, 9:1–38, 2007.

[Haa07] Laura M. Haas. Beauty and the beast: The theory and practice of information integration. In *ICDT*, pages 28–43, 2007.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[HBF$^+$09] Cornelia Hedeler, Khalid Belhajjame, Alvaro A. A. Fernandes, Suzanne M. Embury, and Norman W. Paton. Dimensions of dataspaces. In *BNCOD*, pages 55–66, 2009.

[HBM$^+$] Cornelia Hedeler, Khalid Belhajjame, Lu Mao, Chenjuan Guo, Ian Arundale, Bernadette Farias Lóscio, Norman W. Paton, Alvaro A.A. Fernandes, and Suzanne M. Embury. DSToolkit: An architecture for flexible Dataspace Management. TLDKS Journal (to appear).

[HFM06] Alon Y. Halevy, Michael J. Franklin, and David Maier. Principles of dataspace systems. *PODS*, pages 1–9, 2006.

[HHB10] Jun Hong, Zhongtian He, and David A. Bell. An evidential approach to query interface matching on the deep web. *Inf. Syst.*, 35(2):140–148, 2010.

[HHH+05] Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD Conference*, pages 805–810, 2005.

[HIST03] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation in peer data management systems. In *ICDE*, pages 505–516, 2003.

[HQC08] Wei Hu, Yuzhong Qu, and Gong Cheng. Matching large ontologies: A divide-and-conquer approach. *Data Knowl. Eng.*, 67(1):140–160, 2008.

[HRO06] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data integration: The teenage years. *VLDB*, pages 9–16, 2006.

[IIK08] Aminul Islam, Diana Zaiu Inkpen, and Iluju Kiringa. Applications of corpus-based semantic similarity and word segmentation to database schema matching. *VLDB J.*, 17(5):1293–1320, 2008.

[JFH08] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. *SIGMOD Conference*, pages 847–860, 2008.

[KCGS93] Won Kim, Injun Choi, Sunit K. Gala, and Mark Scheevel. On resolving schematic heterogeneity in multidatabase systems. *Distributed and Parallel Databases*, 1(3):251–279, 1993.

[KN03] Jaewoo Kang and Jeffrey F. Naughton. On schema matching with opaque column names and data values. *SIGMOD Conference*, pages 205–216, 2003.

[KN04] Michael Krauthammer and Goran Nenadic. Term identification in the biomedical literature. *Journal of Biomedical Informatics*, 37(6):512–526, 2004.

[KN08] Jaewoo Kang and Jeffrey F. Naughton. Schema matching using interattribute dependencies. *IEEE Trans. Knowl. Data Eng.*, 20(10):1393–1407, 2008.

[Kol05] Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. *PODS*, pages 61–75, 2005.

[Kot09] Yannis Kotidis. View definition. In *Encyclopedia of Database Systems*, pages 3325–3326. 2009.

[KQ$^+$09] David Kensche, Christoph Quix, Xiang Li 0002, Yong Li, and Matthias Jarke. Generic schema mappings for composition and query answering. *Data Knowl. Eng.*, 68(7):599–621, 2009.

[KQLJ07] David Kensche, Christoph Quix, Yong Li, and Matthias Jarke. Generic schema mappings. In *ER*, pages 132–148, 2007.

[KS91] Won Kim and Jungyun Seo. Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*, 24(12):12–18, 1991.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. *PODS*, pages 233–246, 2002.

[Ler00] Barbara Staudt Lerner. A model for compound type changes encountered in schema evolution. *ACM Trans. Database Syst.*, 25(1):83–127, 2000.

[LEW00] Stephen W. Liddle, David W. Embley, and Scott N. Woodfield. An active, object-oriented, model-equivalent programming language. In *Advances in Object-Oriented Data Modeling*, pages 335–361. 2000.

[LMMS$^+$07] Brenton Louie, Peter Mork, Fernando Martín-Sánchez, Alon Y. Halevy, and Peter Tarczy-Hornoch. Data integration and genomic medicine. *Journal of Biomedical Informatics*, 40(1):5–16, 2007.

[LN07] Frank Legler and Felix Naumann. A classification of schema mappings and analysis of mapping tools. *BTW*, pages 449–464, 2007.

[LSDR07] Yoonkyong Lee, Mayssam Sayyadian, AnHai Doan, and Arnon Rosenthal. etuner: tuning schema matching software using synthetic scenarios. *VLDB J.*, 16(1):97–122, 2007.

[MA10] Hatem A. Mahmoud and Ashraf Aboulnaga. Schema clustering and retrieval for multi-domain pay-as-you-go data integration systems. In *SIGMOD Conference*, pages 411–422, 2010.

[MAB07] Sergey Melnik, Atul Adya, and Philip A. Bernstein. Compiling mappings to bridge applications and databases. In *SIGMOD Conference*, pages 461–472, 2007.

[MBDH05] Jayant Madhavan, Philip A. Bernstein, AnHai Doan, and Alon Y. Halevy. Corpus-based schema matching. *ICDE*, pages 57–68, 2005.

[MBHR05] Sergey Melnik, Philip A. Bernstein, Alon Y. Halevy, and Erhard Rahm. Supporting executable mappings in model management. *SIGMOD Conference*, pages 167–178, 2005.

[MBPF09] Lu Mao, Khalid Belhajjame, Norman W. Paton, and Alvaro A. A. Fernandes. Defining and using schematic correspondences for automatically generating schema mappings. In *CAiSE*, pages 79–93, 2009.

[MBR01] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. *VLDB*, pages 49–58, 2001.

[MCD$^+$07] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. *CIDR*, pages 342–350, 2007.

[MER06] Sabine Massmann, Daniel Engmann, and Erhard Rahm. COMA++: Results for the Ontology Alignment Contest OAEI 2006. *Ontology Matching*, 2006.

[MF04] Zbigniew Michalewicz and David B. Fogel. *How to solve it: modern heuristics*. Springer-Verlag New York Inc, 2004.

[MFH$^+$01] Renée J. Miller, Daniel Fisla, Mary Huang, David Kymlicka, Fei Ku, and Vivian Lee. The Amalgam Schema and Data Integration Test Suite. www.cs.toronto.edu/ miller/amalgam, 2001.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: a versatile graph matching algorithm and itsapplication to schema matching. *ICDE*, pages 117–128, 2002.

[MH03] Jayant Madhavan and Alon Y. Halevy. Composing mappings among data sources. In *VLDB*, pages 572–583, 2003.

[MHH00] Renée J. Miller, Laura M. Haas, and Mauricio A. Hernández. Schema mapping as query discovery. *Proceedings of the 26th International Conference on Very Large Data Bases table of contents*, pages 77–88, 2000.

[Mil95] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[MRB03] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: a programming platform for generic model management. *ACM SIGMOD*, pages 193–204, 2003.

[MSV93] Heinz Mühlenbein and Dirk Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm, i: Continuous parameter optimization. *Evolutionary Computation*, 1(1):25–49, 1993.

[NM01] Natalya F. Noy and Mark A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Proceedings of the workshop on ontologies and information sharing at the international joint conference on artificial intelligence (IJCAI)*, pages 63–70. Citeseer, 2001.

[OV89] M. Tamer Ozsu and Patrick Valduriez. *Principles of distributed database systems.* Addison-wesley Reading Menlo Park, 1989.

[PB08] Rachel Pottinger and Philip A. Bernstein. Schema merging and mapping creation for relational sources. In *EDBT*, pages 73–84, 2008.

[PDYP05] Rong Pan, Zhongli Ding, Yang Yu, and Yun Peng. A bayesian network approach to ontology mapping. In *International Semantic Web Conference*, pages 563–577, 2005.

[PS11] Laura Po and Serena Sorrentino. Automatic generation of probabilistic relationships for improving schema matching. *Inf. Syst.*, 36(2):192–208, 2011.

[PT09] Paolo Papotti and Riccardo Torlone. Schema exchange: Generic mappings for transforming data and metadata. *Data Knowl. Eng.*, 68(7):665–682, 2009.

[PVM$^+$02] Lucian Popa, Yannis Velegrakis, Renée J. Miller, Mauricio A. Hernández, and Ronald Fagin. Translating web data. In *VLDB*, pages 598–609, 2002.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal The International Journal on Very Large Data Bases*, 10(4):334–350, 2001.

[Riz04] Nikos Rizopoulos. Automatic discovery of semantic relationships between schema elements. In *ICEIS (1)*, pages 3–8, 2004.

[SCED89] J. David Schaffer, Rich Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *ICGA*, pages 51–60, 1989.

[SDH08] Anish Das Sarma, Xin Dong, and Alon Y. Halevy. Bootstrapping pay-as-you-go data integration systems. *SIGMOD Conference*, pages 861–874, 2008.

[SDK$^+$07] Marcos Antonio Vaz Salles, Jens-Peter Dittrich, Shant Kirakos Karakashian, Olivier René Girard, and Lukas Blunschi. itrails: Pay-as-you-go information integration in dataspaces. *VLDB*, pages 663–674, 2007.

[SE05] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. *J. Data Semantics IV*, pages 146–171, 2005.

[Sha76] Glenn Shafer. *A mathematical theory of evidence*, volume 1. Princeton university press Princeton, NJ, 1976.

[SKS02] Abraham Silberschatz, Henry Korth, and S. Sudarshan. *Database system concepts*. McGraw-Hill New York, 2002.

[SLDR05] Mayssam Sayyadian, Yoonkyong Lee, AnHai Doan, and Arnon Rosenthal. Tuning schema matching software using synthetic scenarios. pages 994–1005, 2005.

[SMH$^+$10] Len Seligman, Peter Mork, Alon Y. Halevy, Ken Smith, Michael J. Carey, Kuang Chen, Chris Wolf, Jayant Madhavan, Akshay Kannan, and Doug Burdick. Openii: an open source information integration toolkit. In *SIGMOD Conference*, pages 1057–1060, 2010.

[SMM$^+$09] Ken Smith, Michael Morse, Peter Mork, Maya Hao Li, Arnon Rosenthal, David Allen, and Len Seligman. The role of schema matching in large enterprises. In *CIDR*, 2009.

[SWY75] Gerard Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[TC07] Naiyana Tansalarak and Kajal T. Claypool. QMatch–Using paths to match XML schemas. *Data & Knowledge Engineering*, 60(2):260–282, 2007.

[TIP10] Partha Pratim Talukdar, Zachary G. Ives, and Fernando Pereira. Automatically incorporating new sources in keyword search-based data integration. In *SIGMOD Conference*, pages 387–398, 2010.

[TLL$^+$06] Jie Tang, Juan-Zi Li, Bangyong Liang, Xiaotong Huang, Yi Li, and Kehong Wang. Using bayesian decision for ontology mapping. *J. Web Sem.*, 4(4):243–262, 2006.

[Ull00] Jeffrey D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2):189–210, 2000.

[WP08] Ting Wang and Rachel Pottinger. Semap: a generic mapping construction system. In *EDBT*, pages 97–108, 2008.

[WT06] Robert H. Warren and Frank Wm. Tompa. Multi-column substring matching for database schema translation. *VLDB*, pages 331–342, 2006.

[XE06] Li Xu and David W. Embley. A composite approach to automating direct and indirect schema mappings. *Inf. Syst.*, 31(8):697–732, 2006.

[YP04] Cong Yu and Lucian Popa. Constraint-based xml query rewriting for data integration. In *SIGMOD Conference*, pages 371–382, 2004.

[ZLL$^+$09] Qian Zhong, Hanyu Li, Juanzi Li, Guo Tong Xie, Jie Tang, Lizhu Zhou, and Yue Pan. A gauss function based approach for unbalanced ontology matching. In *SIGMOD Conference*, pages 669–680, 2009.

Appendix A

Amalgam Benchmark

The schemas, namely s1, s2, s3 and s4, provided by the Amalgam Benchmark [MFH$^+$01] are included in the appendix.

A.1 Schema s1

The entity-relationship diagram of schema s1 is presented in Figure A.1. The schema definition of s1 is described as follows:

```
create table s1.InProceedings
(inprocID char(35) not null, title char(254), bktitle char(250), year int,
month char(8), pages char(12), vol int, num int, loc char(150),
class char(150), note char(254), annote varchar(2800),
primary key(inprocID))

create table s1.Article
(articleID char(35) not null, title char(254), journal char(150), year int,
month char(8), pages char(12), vol int, num int, loc char(150),
class char(150), note char(254), annote varchar(2800),
primary key(articleID))

create table s1.TechReport
(techID char(35) not null, title char(254), inst char(200), year int,
month char(8), pages char(12), vol int, num int, loc char(150),
class char(150), note char(254), annote varchar(2800),
```

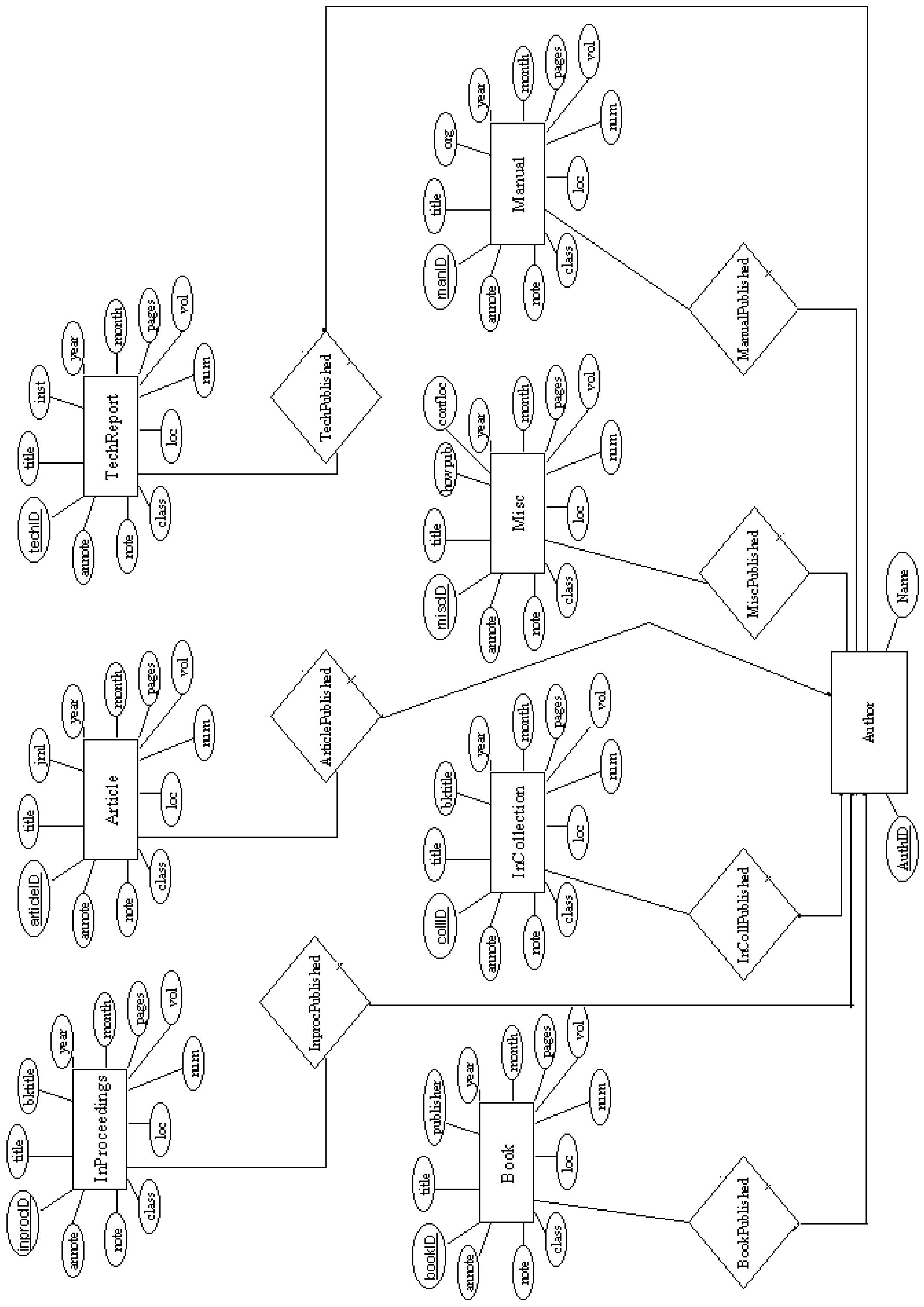


Figure A.1: ER Diagram of Schema s1.

```
primary key(techID))

create table s1.Book
(bookID char(35) not null, title char(254), publisher char(200), year int,
month char(8), pages char(12), vol int, num int, loc char(150),
class char(150), note char(254), annote varchar(2800),
primary key(bookID))

create table s1.InCollection
(collID char(35) not null, title char(254), bktitle char(250), year int,
month char(8), pages char(12), vol int, num int, loc char(150),
class char(150), note char(254), annote varchar(2800),
primary key(collID))

create table s1.Misc
(miscID char(35) not null, title char(254), howpub char(200),
confloc char(100), year int,
month char(8), pages char(12), vol int, num int, loc char(150),
class char(150), note char(254), annote varchar(2800),
primary key(miscID))

create table s1.Manual
(manID char(35) not null, title char(254), org char(200), year int,
month char(8), pages char(12), vol int, num int, loc char(150),
class char(150), note char(254), annote varchar(2800),
primary key(manID))

create table s1.Author
(AuthID int not null, name char(80) not null,
primary key (AuthID))

–RELATIONSHIP TABLES

create table s1.InprocPublished
(inprocID char(35) not null, AuthID int not null,
primary key(inprocID, AuthID),
foreign key (inprocID) references s1.InProceedings
on delete CASCADE, foreign key (AuthID)
references s1.Author on delete CASCADE)

create table s1.ArticlePublished
(articleID char(35) not null, AuthID int not null,
primary key(articleID, AuthID),
foreign key (articleID) references s1.Article
on delete CASCADE, foreign key (AuthID)
references s1.Author on delete CASCADE)

create table s1.TechPublished
(techID char(35) not null, AuthID int not null,
primary key(techID, AuthID),
foreign key (techID) references s1.TechReport
on delete CASCADE, foreign key (AuthID)
references s1.Author on delete CASCADE)

create table s1.BookPublished
(bookID char(35) not null, AuthID int not null,
primary key(bookID, AuthID),
foreign key (bookID) references s1.Book
on delete CASCADE, foreign key (AuthID)
references s1.Author on delete CASCADE)

create table s1.InCollPublished
(collID char(35) not null, AuthID int not null,
primary key(collID, AuthID),
foreign key (collID) references s1.InCollection
on delete CASCADE, foreign key (AuthID)
references s1.Author on delete CASCADE)

create table s1.MiscPublished
(miscID char(35) not null, AuthID int not null,
primary key(miscID, AuthID),
foreign key (miscID) references s1.Misc
on delete CASCADE, foreign key (AuthID)
references s1.Author on delete CASCADE)

create table s1.ManualPublished
(manID char(35) not null, AuthID int not null,
primary key (manID, AuthID),
foreign key (manID) references s1.Manual
on delete CASCADE, foreign key (AuthID)
references s1.Author on delete CASCADE)
```

A.2 Schema s2

The entity-relationship diagram of schema s2 is presented in Figure A.2. The schema definition of s2 is described as follows:

```
create table s2.allBibs
( citKey char(20) not null, primary key (citKey) )

create table s2.citForm
( citKey char(20) not null, form char(20),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.authors
( citKey char(20) not null, autNm char(100) not null,
primary key (citKey, autNm),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.editors
( citKey char(20) not null,
edNm char(100) not null, primary key (citKey, edNm),
foreign key (citKey) references s2.allBibs on delete cascade )
```

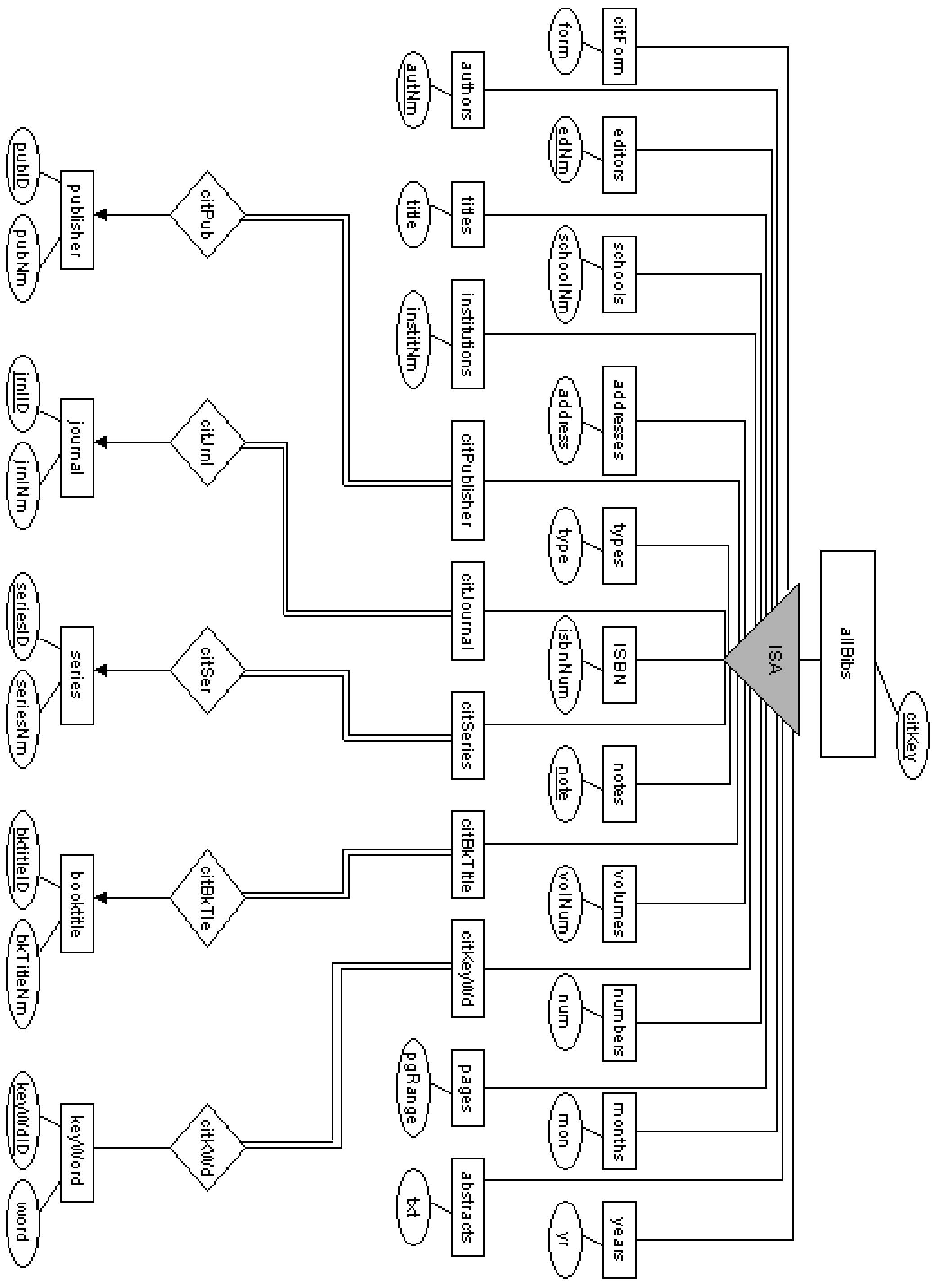


Figure A.2: ER Diagram of Schema s2.

```
create table s2.titles
( citKey char(20) not null, title char(200),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.schools  ( citKey char(20) not null, schoolNm char(100),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.institutions
( citKey char(20) not null, institNm char(100),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.addresses
( citKey char(20) not null, address char(100),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.types
( citKey char(20) not null, type char(100),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.ISBN
( citKey char(20) not null, isbnNum char(20),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.notes
( citKey char(20) not null, note char(200) not null,
primary key (citKey, note),
foreign key (citKey) references s2.allBibs on delete cascade )
```

```
create table s2.volumes
( citKey char(20) not null, volNum char(50),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.numbers
( citKey char(20) not null, num char(50),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.months
( citKey char(20) not null, mon char(20),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.years
( citKey char(20) not null, yr char(20),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.pages
( citKey char(20) not null, pgRange char(50),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.abstracts
( citKey char(20) not null, txt varchar(3500),
primary key (citKey),
foreign key (citKey) references s2.allBibs on delete cascade )

create table s2.publisher
( pubNm char(100), pubID integer not null,
primary key (pubID) )

create table s2.citPublisher
```

```
( citKey char(20) not null, pubID integer not null,
primary key (citKey, pubID),
foreign key (citKey) references s2.allBibs on delete cascade,
foreign key (pubID) references s2.publisher on delete cascade )

create table s2.journal
( jrnlNm char(200), jrnlID integer not null,
primary key (jrnlID) )

create table s2.citJournal
( citKey char(20) not null, jrnlID integer not null,
primary key (citKey, jrnlID),
foreign key (citKey) references s2.allBibs on delete cascade,
foreign key (jrnlID) references s2.journal on delete cascade )

create table s2.series
( seriesNm char(50), seriesID integer not null,
primary key (seriesID) )

create table s2.citSeries
( citKey char(20) not null, seriesID integer not null,
primary key (citKey, seriesID),
foreign key (citKey) references s2.allBibs on delete cascade,
foreign key (seriesID) references s2.series on delete cascade )

create table s2.booktitle ( bkTitleNm char(100), bktitleID integer not null,
primary key (bktitleID) )

create table s2.citBkTitle
( citKey char(20) not null, bktitleID integer not null,
primary key (citKey, bktitleID),
foreign key (citKey) references s2.allBibs on delete cascade,
foreign key (bktitleID) references s2.booktitle on delete cascade )

create table s2.keyWord
```

```
( word char(50), keyWdID integer not null,
primary key (keyWdID) )

create table s2.citKeyWd
( citKey char(20) not null, keyWdID integer not null,
primary key (citKey, keyWdID),
foreign key (citKey) references s2.allBibs on delete cascade,
foreign key (keyWdID) references s2.keyWord on delete cascade )
```

A.3 Schema s3

The entity-relationship diagram of schema s3 is presented in Figure A.3. The schema definition of s3 is described as follows:

```
create table s3.article (
articleID varchar(30) not null,
title varchar(150) not null,
volume int not null,
number varchar(20)not null,
pages varchar(30) not null,
month varchar(100) not null,
year int not null,
refkey varchar(50),
note varchar(150),
remarks varchar(400),
references varchar(2000),
xxxreferences varchar(600),
fullxxxreferences varchar(400),
oldkey varchar(50),
abstract varchar(3000),
preliminary varchar(100),
primary key (articleID)) IN CSC494TABLESPACE

create table s3.author (
```

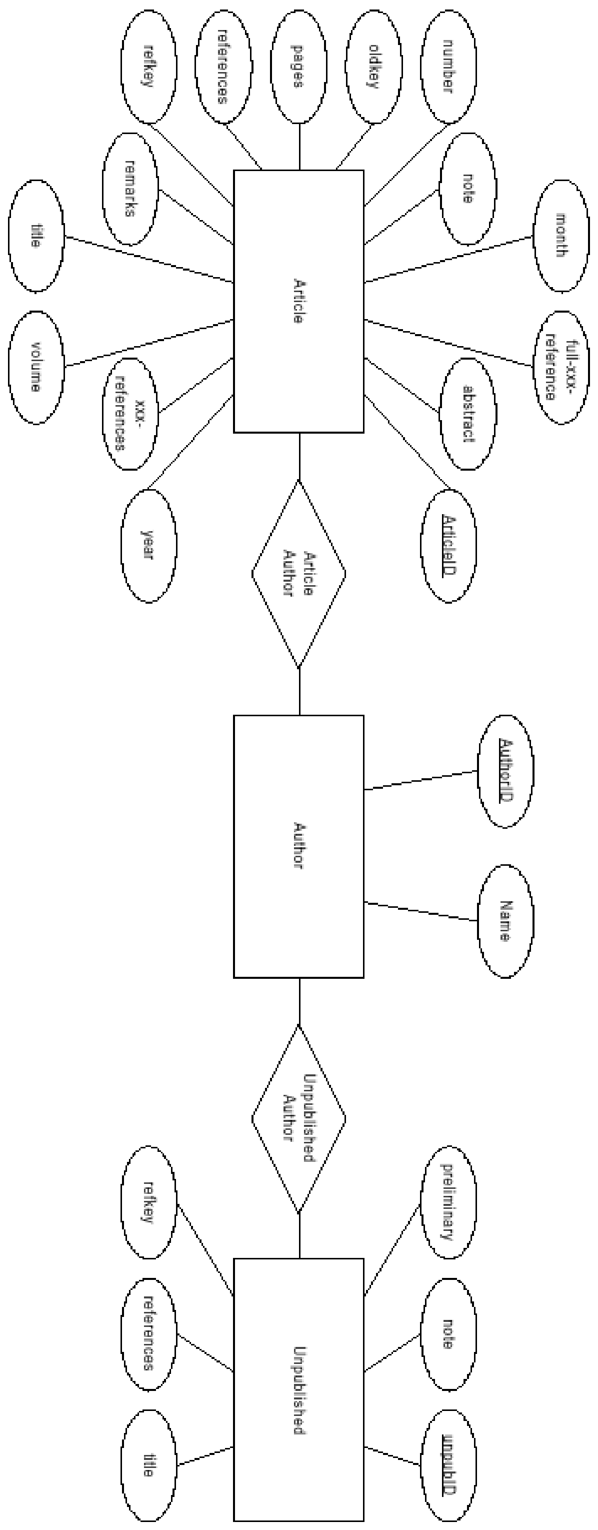


Figure A.3: ER Diagram of Schema s3.

```
authorID int not null,
name varchar(40) not null,
primary key (authorID))


create table s3.unpublished (
unpubID varchar(30) not null,
title varchar(150) not null,
refkey varchar(20),
note varchar(50),
preliminary varchar(100),
references varchar(2000),
primary key(unpubID))


create table s3.articleAuthor (
articleID varchar(30) not null,
authorID int not null,
primary key (articleID, authorID),
foreign key (articleID) references s3.article
on delete cascade,
foreign key (authorID) references s3.author
on delete restrict)


create table s3.unpubAuthor (
unpubID varchar(30) not null,
authorID int not null,
primary key (unpubID, authorID),
foreign key (unpubID) references s3.unpublished
on delete cascade,
foreign key (authorID) references s3.author
on delete restrict)
```

A.4 Schema s4

The entity-relationship diagram of schema s4 is presented in Figure A.4. The schema definition of s4 is described as follows:

```
create table s4.author (
aid integer not null, name varchar(50) not null unique,
affiliations varchar(200),
primary key(aid))

create table s4.descriptor (
did integer not null, descriptor varchar(50) not null unique,
primary key(did) )

create table s4.location (
lid integer not null , countrypub varchar(50) not null,
countryorigin varchar(50) not null,
unique(countrypub,countryorigin),
primary key(lid) )

create table s4.publication (
pid integer not null, title varchar(500) not null,
titleext varchar(100), abstract varchar(3000) not null,
abstractind varchar(5) not null, language varchar(50) not null,
journal varchar(255), journalann varchar(20), confinfo varchar(500),
book varchar(500), category varchar(100) not null,
primary key (pid) )

create table s4.record (
rid integer not null, availability varchar(255) not null,
updatecode varchar(25) not null, numref varchar(3),
contractnum varchar(100), issn varchar(25),
isbn varchar(25), notes varchar(100),
subfile varchar(10) not null, source varchar(255),
series varchar(100), accessionnum varchar(100) not null,
primary key (rid) )
```

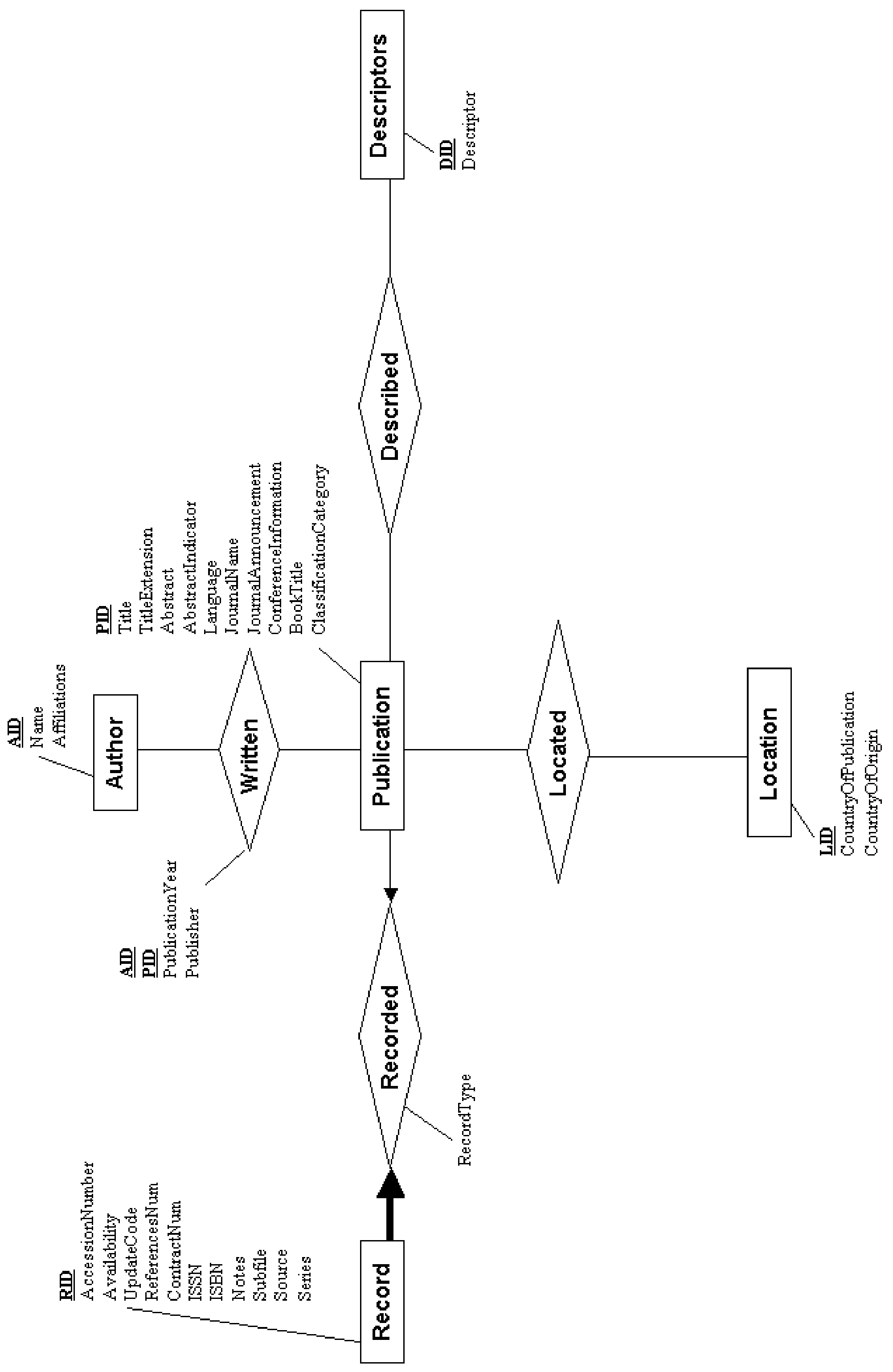


Figure A.4: ER Diagram of Schema s4.

```
create table s4.described (
pid integer not null, did integer not null,
primary key(pid, did),
foreign key(pid) references s4.publication,
foreign key(did) references s4.descriptor)

create table s4.located (
pid integer not null, lid integer not null,
primary key(pid, lid),
foreign key(pid) references s4.publication,
foreign key(lid) references s4.location)

create table s4.recorded (
pid integer not null, rid integer not null, recordtype varchar(100),
primary key(pid, rid),
foreign key(pid) references s4.publication,
foreign key(rid) references s4.record)

create table s4.written (
aid integer not null, pid integer not null,
pubyear varchar(100) not null, publisher varchar(100),
primary key(pid, aid),
foreign key(pid) references s4.publication,
foreign key(aid) references s4.author)
```