# Chapter 1

# INTRODUCTION

This dissertation studies *representation matching*: the problem of creating semantic mappings between two data representations. Examples of mappings are "element location of one representation maps to element address of the other", "contact-phone maps to agent-phone", and "listed-price maps to price * (1 + tax-rate)".

We begin this chapter by showing that representation matching is a fundamental step in numerous data management applications. Next, we show that the manual creation of semantic mappings is extremely labor intensive and hence has become a key bottleneck hindering the widespread deployment of the above applications (Sections 1.2-1.3). We then outline our semi-automatic solutions to representation matching (Sections 1.4-1.5). Finally, we list the contributions and give a road map to the rest of the dissertation (Section 1.6-1.7).

## *1.1 Applications of Representation Matching*

The key commonalities underlying applications that require semantic mappings are that they use structured representations (e.g., relational schemas, ontologies, and XML DTDs) to encode the data, and that they employ more than one representation. As such, the applications must establish semantic mappings among the representations, either to enable their manipulation (e.g., merging them or computing the differences [BLN86, BHP00]) or to enable the translation of data and queries across the representations. Many such applications have arisen over time and have been studied actively by the database and AI communities.

One of the earliest such applications is *schema integration*: merging a set of given schemas into a single global schema [BLN86, EP90, SL90, PS98]. This problem has been studied since the early 1980s. It arises in building a database system that comprises several distinct databases, and in designing the schema of a database from the local schemas supplied by several user groups. The integration process requires establishing semantic mappings between the component schemas [BLN86].

As databases become widely used, there is a growing need to *translate data* between multiple databases. This problem arises when organizations consolidate their databases and hence must transfer data from old databases to the new ones. It forms a critical step in *data warehousing* and *data mining*, two important research and commercial areas since the early 1990s. In these applications, data coming from multiple sources must be transformed to data conforming to a single target schema, to enable further data analysis [MHH00, RB01].

During the late 1980s and the 1990s, applications of representation matching arose in the context of *knowledge base construction*, which is studied in the AI community. Knowledge bases store complex types of entities and relationships, using "extended database schemas" called *ontolo-*

*gies* [BKD+01, HH01, Ome01, MS01, iee01]. As with databases, there is a strong need to build new knowledge bases from several component ones, and to translate data among multiple knowledge bases. These tasks require solving the *ontology matching* problem: find semantic mappings between the involved ontologies.

In the recent years, the explosive growth of information online has given rise to even more application classes that require representation matching. One application class builds *data integration systems* [GMPQ+97, LRO96, IFF+99, LKG99, FW97, KMA+98]. Such a system provides users with a *uniform query interface* to a multitude of data sources. The system provides this interface by enabling users to pose queries against a *mediated schema*, which is a virtual schema that captures the domain's salient aspects. To answer queries, the system uses a set of *semantic mappings* between the mediated schema and the local schemas of the data sources, in order to reformulate a user query into a set of queries on the data sources. A critical problem in building a data-integration system, therefore, is to supply the set of semantic mappings between the mediated- and source schemas.

Another important application class is *peer data management*, which is a natural extension of data integration. A peer data management system does away with the notion of mediated schema and allows peers (i.e., participating data sources) to query and retrieve data directly from each other. Such querying and data retrieval require the creation of semantic mappings among the peers.

Recently there has also been considerable attention on *model management*, which creates tools for easily manipulating models of data (e.g., data representations, website structures, and ER diagrams). Here matching has been shown to be one of the central operations [BHP00, RB01].

The data sharing applications described above arise in numerous real-world domains. Applications in databases have now permeated all areas of our life. Knowledge base applications are often deployed in diverse domains such as medicine, commerce, and military. These applications also play important roles in emerging domains such as e-commerce, bioinformatics, and ubiquitous computing [UDB, MHTH01, ILM+00].

Some recent developments should dramatically increase the need for and the deployment of applications that require mappings. The Internet has brought together millions of data sources and makes possible data sharing among them. The widespread adoption of XML as a standard *syntax* to share data has further streamlined and eased the data sharing process. Finally, the vision of the Semantic Web is to publish data (e.g., by marking up webpages) using ontologies, thus making data that is available on the Internet become even more structured. The growth of the Semantic Web will further fuel data sharing applications and underscore the key role that representation matching plays in their deployment.

Representation matching therefore is truly pervasive. Variations of this problem have been referred to in the literature as *schema matching, ontology matching, ontology alignment, schema reconciliation, mapping discovery, reconciling representations, matching XML DTDs*, and *finding semantic correspondences*.

### *1.2 Challenges of Representation Matching*

Despite its pervasiveness and importance, representation matching remains a very difficult problem. Matching two representations $S$ and $T$ requires deciding if any two elements $s$ of $S$ and $t$ of $T$ *match*, that is, if they refer to the same real-world concept. This problem is challenging for several fundamental reasons:

- The semantics of the involved elements can be inferred from only a few information sources, typically the creators of data, documentation, and associated representation schema and data.

  Extracting semantics information from data creators and documentation is often extremely cumbersome. Frequently, the data creators have long moved, retired, or forgotten about the data. Documentation tends to be sketchy, incorrect, and outdated. In many settings such as when building data integration systems over remote Web sources, data creators and documentation are simply not accessible.

- Hence representation elements are typically matched based on clues in the *schema* and *data*. Examples of such clues include element names, types, data values, schema structures, and integrity constraints. However, these clues are often unreliable. For example, two elements that share the same name (e.g., area) can refer to different real-world entities (the location and square-feet area of the house in this case). The reverse problem also often holds: two elements with different names (e.g., area and location) can actually refer to the same real-world entity (the location of the house).

- The above clues are also often incomplete. For example, the name contact-agent only suggests that the element is related to the agent. It does not provide sufficient information to determine the exact nature of the relationship (e.g., whether the element is about the agent's phone number or her name).

- To decide that element $s$ of representation $S$ matches element $t$ of representation $T$, one must typically examine *all* other elements of $T$ to make sure there is no other element that matches $s$ better than $t$. This *global* nature of matching adds substantial cost to the matching process.

- To make matters worse, matching is often *subjective*, depending on the application. One application may decide that house-style matches house-description, another application may decide not. Hence, the user must often be involved in the matching process. Sometimes, even the input of a single user is considered too subjective, and then a whole committee must be assembled to decide on the correct matching [CHR97].

Because of the above challenges, the manual creation of semantic mappings has long been known to be extremely laborious and error-prone. For example, a recent project at the GTE telecommunications company sought to integrate 40 databases that have a total of 27,000 elements (i.e., attributes of relational tables) [LC00]. The project planners estimated that, without the database creators, just finding and documenting the semantic mappings among the elements would take more than 12 person years.

### *1.3 State of the Art*

The high cost of manual mapping has spurred numerous solutions, which fall roughly into two groups:

- The first group *develops standards* (i.e., common vocabularies) that all representations must conform to. This approach eliminates the need for representation matching.

  Standardization can work well for some narrowly defined domains, such as certain business areas. However, it cannot be a general solution to reconciling representations, for several reasons. First, often a domain generates multiple competing standards; this defeats the purpose of having a standard in the first place. Second, as their needs evolve, organizations must extend standards to handle unanticipated data. Extensions from different organizations are generally incompatible with each other. Third, developing standards demands consensus and takes time. This poses a serious problem for newly emerging domains.

  But most importantly, in numerous domains there is always a need to deal with data *originally created for a different purpose*. For example, in data integration, data at the sources is created independently, typically well before the need for integration arises. Such data by nature does not conform to a single domain standard. Hence, the need for representation matching will always remain.

- Since the representation matching problem does not go away, the second solution group seeks to *automate the mapping process*. Because the users must be in the loop, only semi-automatic methods can be considered. Numerous such methods have been developed, in the areas of databases, AI, e-commerce, and the Semantic Web (e.g., [MZ98, PSU98, CA99, LC00, PE95, CHR97, MBR01, MMGR02, MHH00, DR02, Cha00, MFRW00, NM00, MWJ, NM01, RHS01]; see [RB01] for an excellent survey of automatic approaches developed by the database community).

  The proposed approaches have built efficient specialized mapping strategies, and significantly advanced our understanding of representation matching. However, these approaches suffer from two serious shortcomings. First, they typically employ a single matching strategy that exploits only certain types of information and that is often tuned to only certain types of applications. As a result, the solutions have limited applicability and matching accuracy. In particular, they lack modularity and extensibility, and do not generalize well across application domains and data representations. Second, most proposed solutions can discover only 1-1 semantic mappings. They cannot find complex mappings such as "name = concat(first-name, last-name)". This is a serious limitation because complex mappings make up a significant portion of semantic mappings in practice (see Chapter 4 for more detail).

Because no satisfactory solution to representation matching exists, today the vast majority of semantic mappings is still created manually. The slow and expensive manual acquisition of mappings has now become a serious bottleneck in building information processing applications. This problem will become even more critical as data-sharing applications *proliferate* and *scale up*. As mentioned in Section 1.1, the development of technologies such as the Internet, XML, and the Semantic Web will further fuel data-sharing applications, and enable applications to share data across thousands or millions of sources. Manual mapping is simply not possible at such scales.

Hence, the development of *semi-automatic* solutions to representation matching is now truly crucial to building a broad range of information processing applications. Given that representation

matching is a fundamental step in numerous such applications, it is important that the solutions be robust and applicable across domains. This dissertation develops such solutions.

### *1.4 Goals of the Dissertation*

The central thesis of this dissertation is that, *for the representation matching problem, we can design a semi-automatic solution that builds on well-founded semantics, that is broadly applicable, and that exploits multiple types of information and techniques to maximize mapping accuracy.*

Specifically, our goals are as follows:

- Develop a *formal framework* for representation matching. The framework should define relevant notions in representation matching, such as semantic mapping, domain constraints, and user feedback. It should explain the behavior of both system and user, and expose the informal assumptions often made by matching solutions.

- Within the above formal framework, develop a solution that has *broad applicability*: the solution should handle a variety of data representations, such as relational tables, XML DTDs, and ontologies, and should discover both 1-1 and complex semantic mappings.

- Design the solution such that it *maximizes matching accuracy* by exploiting a wide range of information. First, whenever possible, the solution should exploit *previous matching activities*: it should be able to "look over the user's shoulder" to learn how to perform mapping, then propose new mappings itself. Second, any single type of syntactic clue is often unreliable, as we have shown in Section 1.2. Hence, the solution should exploit *multiple types of clues*, to achieve high matching accuracy. Third, the solution should utilize *integrity constraints* that appear frequently in application domains. Finally, because the user must be in the loop, the solution should be able to efficiently incorporate user feedback into the matching process.

To achieve these goals, we proceed with the following steps:

1. Develop a formal framework for representation matching (Chapter 2).

2. Develop and evaluate a solution for discovering 1-1 mappings in the context of data integration, for XML data. Design the solution such that it can exploit multiple types of information (Chapter 3).

3. Extend the solution to discovering complex semantic mappings. Evaluate the solution in the context of matching relational representations for data translation (Chapter 4).

4. Extend the solution to finding 1-1 mappings for ontologies, a representation that is more complex than relational and XML ones (Chapter 5).

### *1.5 Overview of the Solutions*

We now outline our solutions to the above problem steps.

### *1.5.1 Formalizing Representation Matching*

In this dissertation we shall consider several types of matching, all of which require solving the following fundamental problem: *given two representations S and T, for each element s of S, find the most similar element t of T, utilizing all available information.* This includes any information about the representations and their domains, such as data instances, integrity constraints, previous matchings, and user feedback.

The solutions that we shall develop for the above problem compute for each element pair $s \in S$ and $t \in T$ a numeric value $v$. The value $v$ indicates the degree of similarity between $s$ and $t$; the higher $v$ is, the more similar $s$ and $t$ are. We refer to a tuple $(s,t,v)$ as a *semantic mapping*. Then for each element $s \in S$, the solutions return the semantic mapping that involves $s$ and has the highest similarity value $v$.

Many works on representation matching [MBR01, BM01, DR02, CA99, BCVB01, LC94, CHR97, RHS01, LG01] have also considered the above problem and solution approach. However, most of them do not define the problem *formally*. They do not make clear what "most similar element" means, nor do they state the implicit assumptions that underlie their solutions.

We believe a formal framework for representation matching is important because it facilitates the evaluation of solutions. It also makes clear to the users what a solution means by a match, and helps them evaluate the applicability of a solution to a given matching scenario. Furthermore, the formalization may allow us to leverage special-purpose techniques for the matching process. An important contribution of this dissertation is that we developed such a framework, which defines the matching problem and explains the solutions that we develop.

Our framework assumes that the user knows a conceptualization of the domain in terms of a representation $\mathcal{U}$, and that he or she knows a similarity measure $\mathcal{S}$ that is defined over the concepts in $\mathcal{U}$. The framework further assumes that the user can map any two concepts $s$ and $t$ of the input representations into semantically equivalent concepts $\mathcal{M}(s)$ and $\mathcal{M}(t)$ in $\mathcal{U}$. In Chapter 2 we discuss the motivations leading to these assumptions.

Given the above assumptions, we can formally state the matching problem as follows: for each $s$ of $S$, find $t$ that maximizes $\mathcal{S}(\mathcal{M}(\mathrm{s}),\mathcal{M}(\mathrm{t}))$ among all elements of $T$ – in other words, find $t$ such that the similarity value computed by $\mathcal{S}$ between the equivalent concepts in $\mathcal{U}$ is the highest. Thus, when a solution produces a semantic mapping $(s,t,v)$, we can interpret $v$ to be the best estimation of the true similarity value $\mathcal{S}(\mathcal{M}(s),\mathcal{M}(t))$ that the solution could produce.

Notice that in estimating the true similarity values, the solution often has only *partial* knowledge about the representations and their domains. It may know the data types, structures, names, and some data instances associated with representation elements, as well as a set of integrity constraints and some knowledge about $\mathcal{S}$ and $\mathcal{U}$. It must utilize such knowledge and rely largely on the similarities of *syntactic clues* (e.g., element names and data instances) to estimate *semantic similarities* (as represented by $\mathcal{S}$). However, this estimation makes sense only under the assumption that the syntactic similarity is positively and strongly correlated with the semantic similarity. This assumption is frequently made but rarely stated explicitly by previous works in representation matching.

Our framework thus explains that what matching solutions attempt to do is to estimate true similarity values as represented by $\mathcal{S}$. In the next three sections, we study *how* to obtain *good* estimates of true similarity values, in the context of specific matching problems.

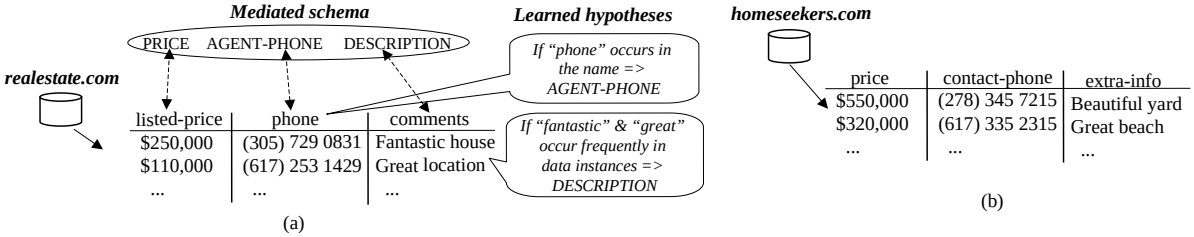


Figure 1.1: Once LSD has trained a set of learners on source *realestate.com* in (a), it can apply the learners to find semantic mappings for source *homeseekers.com* in (b).

### *1.5.2 1-1 Matching for Data Integration*

We begin by considering the basic 1-1 matching problem that is described in the previous section, but in the context of data integration systems. We choose data integration because it is an important data management application and because it provides a general problem setting, the solution of which could be generalized to other applications such as data translation and ontology matching (Chapters 4-5).

Recall from Section 1.1 that a data integration system enables users to retrieve data from a multitude of sources by posing queries on a *mediated schema*. To answer queries, the system must know the semantic mappings between the mediated schema and the schemas of the data sources. Our goal is to develop a solution to semi-automatically create these mappings.

Below we briefly describe the solution as embodied in the LSD system that we have developed. The key idea underlying LSD is that after the schemas of a few data sources have been manually mapped to the mediated schema, it should be able to learn from the manual mappings to successfully propose mappings for subsequent data sources.

**Example 1** Consider a data-integration system that helps users find houses on the real-estate market. Suppose that the system has the mediated schema shown in Figure 1.1.a. The mediated schema (which is a simplification of a real one) consists of three elements: PRICE, AGENT-PHONE, and DESCRIPTION.

Suppose further that we have selected source *realestate.com* and manually specified the 1-1 mappings between the schema of this source and the mediated schema. This amounts to specifying the three dotted arrows in Figure 1.1.a. The first arrow, for example, states that source-schema element listed-price matches mediated-schema element PRICE. (We use THIS FONT and this font to refer to the elements of mediated and source schemas, respectively.)

Once we have specified the mappings, there are many different types of information that LSD can glean from the source schema and data to train a set of *learners*. A learner can exploit the *names* of schema elements: knowing that phone matches AGENT-PHONE, it could hypothesize that if an element name contains the word "phone", then that element is likely to be AGENT-PHONE. The learner can also look at example phone numbers in the source data, and learn the *format* of phone numbers. It could also learn from *word frequencies*: it could discover that words such as "fantastic" and "great" appear frequently in house descriptions. Hence, it may hypothesize that if these words appear frequently in the data instances of an element, then that element is likely to be

DESCRIPTION. As yet another example, the learner could also learn from the *characteristics of value distributions*: it can look at the average value of an element, and learn that if that value is in the thousands, then the element is more likely to be price than the number of bathrooms.

Once the learners have been trained, we apply LSD to find semantic mappings for new data sources. Consider source *homeseekers.com* in Figure 1.1.b. A word-frequency learner may examine the word frequencies of the data instances of element extra-info, and recognize that these data instances are house descriptions. Based on these predictions, LSD will be able to predict that extra-info matches DESCRIPTION.□

As described, machine learning provides an attractive platform for finding semantic mappings. However, applying it to our domain raises several challenges. The first challenge is to decide which learners to employ in the training phase. A plethora of learning algorithms have been described in the literature, each of which has strengths in learning different types of patterns. A key distinguishing factor of LSD is that we take a *multi-strategy learning* approach [MT94]: we employ a multitude of learners, called *base learners*, then combine the learners' predictions using a *meta-learner*. An important feature of multi-strategy learning is that our system is *extensible* since we can add new learners that have specific strengths in particular domains, as these learners become available.

The second challenge is to exploit integrity constraints that appear frequently in database schemas and to incorporate user feedback on the proposed mappings in order to improve accuracy. We extended multi-strategy learning to incorporate these. As an example of exploiting integrity constraints, suppose we are given a constraint stating that the value of the mediated-schema element HOUSE-ID is a key for a real-estate entry. In this case, LSD would know not to match num-bedrooms to HOUSE-ID because the data values of num-bedrooms contain duplicates, and thus it cannot be a key. As an example of incorporating user feedback, LSD can benefit from feedback such as "ad-id does not match HOUSE-ID" to constrain the mappings it proposes.

The third challenge arises from the nature of XML data. We built LSD to match both relational and XML data. However, while experimenting with LSD, we realized that none of its learners could handle the hierarchical structure of XML data very well (see Chapter 3). Hence, we developed a novel learner, called the *XML learner*, that handles hierarchical structure and further improves the accuracy of our mappings.

We evaluated LSD on several real-world data integration domains. The results show that with the current set of learners, LSD already obtains predictive accuracy of 71-92% across all domains. The experiments show the utility of multi-strategy learning and of exploiting domain constraints and user feedback for representation matching.

### *1.5.3 Complex Matching*

LSD provides a powerful matching solution that can exploit multiple types of information. However, it discovers only 1-1 semantic mappings such as "DESCRIPTION = comments". Since complex mappings such as "NUM-BATHS = full-baths + half-baths" and "ADDRESS = *concat*(city,zipcode)" are also widespread in practice, we developed the COMAP system, which extends LSD to find both 1-1 and complex mappings.

We explain COMAP using the familiar data-integration setting. Here, for each mediated-schema element $s$, COMAP considers finding the best semantic mapping (be it 1-1 or complex) over the elements of a given source schema $T$. To do this, COMAP *quickly* finds a small set of best candidate

mappings for $s$. Next, it "adds" the newly found mappings to source schema $T$ so that they can be treated as additional "composite" elements of $T$. For instance, suppose $T$ consists of three elements price, city, and state. Suppose further that the best candidate mappings for mediated-schema element ADDRESS are $\{concat(\text{city}, \text{state}), concat(\text{price}, \text{city})\}$. Then $concat(\text{city}, \text{state})$ would be a new "composite" element of $T$, whose data instances would be obtained by concatenating those of city and state.

Once candidate mappings for all mediated-schema elements have been computed and added to source schema $T$, GLUE applies LSD (which is modified to fit the complex-matching context) to find 1-1 semantic mappings between the mediated schema and the expanded schema of $T$. Continuing with the above ADDRESS example, if LSD matches ADDRESS with the composite element that corresponds to the candidate $concat(\text{city}, \text{state})$, then GLUE would return this candidate as the best mapping for ADDRESS.

As described, reducing complex matching to 1-1 matching provides an elegant framework that can utilize all techniques previously developed for 1-1 matching (including our LSD work). However, it raises several challenges. The first challenge is how to efficiently search the vast (often infinite) space of all 1-1 and complex mappings, to find the best candidate mappings. COMAP solves this problem by *breaking up* the search space: it employs multiple *searchers* – each exploits a certain type of information to quickly find a small set of promising candidate mappings – then returns the union of the mappings found by all searchers. Multisearch therefore is a natural extension of multistrategy learning employed in LSD. As such, it provides COMAP with a high degree of modularity and extensibility.

The second challenge is how to implement the searchers and evaluate candidate mappings. COMAP provides a default implementation using *beam search*, and uses machine learning and statistical techniques to evaluate candidate mappings. Naturally, searchers can choose not to use the default implementation if a more suitable technique becomes available.

In Chapter 4 we provide a detailed description of COMAP, as well as of the experiments we conducted with it on real-world data. There, we explain its implementation for matching relational data and the necessary changes to extend it to matching XML data.

### *1.5.4 Ontology Matching*

Together, LSD and COMAP provide a solution that covers both 1-1 and complex matching. We must extend this solution in two important aspects. First, the solution matches only relational and XML representations; we extend it to also match ontologies. Ontologies have proven very popular as representations of data. They play a key role in constructing knowledge bases and marking up data on the proposed Semantic Web [BKD+01, BLHL01]. Hence, ontology matching should be an integral part of any general matching solution.

Second, the solution we developed can exploit a broad range of information, including schema and data information, integrity constraints, past matchings, and user feedback. However, it has not considered exploiting any information that is available about the similarity measure defined over representation elements (i.e., measure $\mathcal{S}$ defined in Section 1.5.1). In many practical settings, the user does know the similarity measure and can supply it as a problem input. Hence, we consider such settings and extend our solution to exploit user-supplied similarity measures, to improve the estimation of true similarity values.

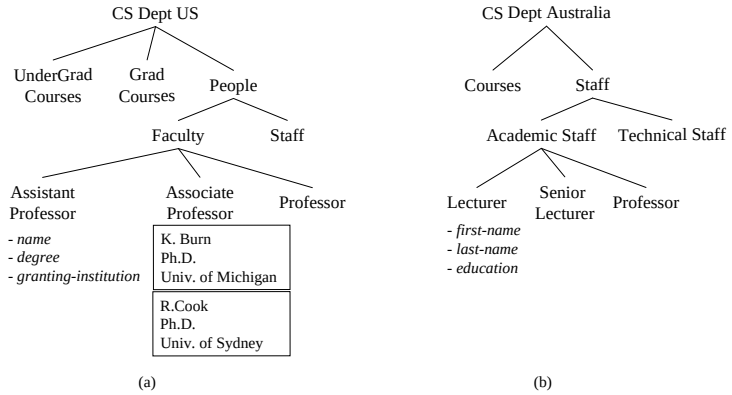


Figure 1.2: Sample ontologies in the CS department domain

We have developed the GLUE system, which extended LSD and COMAP in the two ways described above. The current GLUE focuses on matching *taxonomies*, which are central components of ontologies. A taxonomy is a tree where each node represents a *concept* and each concept is a specialization of its parent. Figure 1.2 shows two sample taxonomies for the CS department domain. Given two taxonomies and a user-defined similarity measure, GLUE finds for each concept node in a taxonomy the most similar concept node in the other taxonomy.

The first challenge GLUE faces is how to compute the similarity of any two concepts in the taxonomies. A key observation we made is that many practical similarity measures can be defined based solely on the *joint probability distribution* of the concepts involved. For example, the well-known *Jaccard* measure [vR79] computes the similarity between two concepts $A$ and $B$ to be $P(A \cap B)/P(A \cup B)$, which can be re-expressed in terms of the joint distribution of $A$ and $B$.

GLUE assumes that the user-supplied similarity measure also has the above property. Then, instead of attempting to estimate specific similarity values directly, GLUE focuses on computing the joint distributions. After that, it is possible to compute any similarity measure such as the *Jaccard* coefficient as a function of the joint distributions. GLUE therefore has the significant advantage of being able to work with a variety of similarity functions. We apply multistrategy learning as described in LSD to compute the joint distributions of the concepts. We describe this process in detail in Chapter 5.

The second challenge for GLUE is that the taxonomy structure gives rise to matching heuristics that have not been considered in the context of relational and XML data. An example heuristic is that two nodes are likely to match if their parents and descendants also match. Such heuristics occur frequently in practice, and are very commonly used when manually mapping between ontologies. Previous works have exploited only one form or another of such knowledge, in restrictive settings [NM01, MZ98, MBR01, MMGR02].

In the GLUE context we developed a unifying approach to incorporate all such types of heuristic. Our approach is based on *relaxation labeling*, a powerful technique that has been used successfully in vision and image processing [HZ83], natural language processing [Pad98], and hypertext classification [CDI98]. We show that relaxation labeling can be adapted efficiently to our context, and that it can successfully handle a broad variety of heuristics. Chapter 5 describes relaxation labeling

and the rest of GLUE in detail. It also describes experiments we conducted on real-world domains to validate GLUE.

### *1.6 Contributions of the Dissertation*

Up to the time of this dissertation, most works have employed only hand-crafted rules to match representations. Several recent works have advocated the use of learning techniques (see Chapter 6 on the related work). However, in general it is not clear how to reconcile the two approaches. There is also an implicit and gradual realization that multiple types of information must be exploited to maximize matching accuracy. However, it is not clear what is a good way to exploit them and combine their effects. Finally, the vast majority of works have considered only 1-1 matching. It is not clear what is a good way to attack the problem of complex matching, and whether a solution that unifies both 1-1 and complex matching can be developed.

- The most important contribution of this dissertation is a solution architecture that provides answers to the above questions. The solution advocates the use of *multiple independent modules*, each exploiting a certain type of information. Meta-learning techniques then utilize training data to find the best way to combine module predictions. As such, the solution provides a unifying framework for previous approaches: its modules can employ rules, learning techniques, or any other techniques deemed most suitable for exploiting the information at hand. The multi-module nature also makes the solution easily extensible and customized to any particular application domain.

  The solution combines both 1-1 and complex matching. Furthermore, it provides a unifying and efficient approach to incorporate a broad range of integrity constraints and domain heuristics. It can utilize previous matching activities and incorporate user feedback, as we show with LSD. It can handle a variety of representations, including relational, XML, and ontologies. Finally, it can also handle a broad range of similarity measures, an ability that is missing from most previous matching solutions.

- Another major contribution of the dissertation is a semantics framework that formally defines representation matching. The framework explains the solutions commonly adopted in practice, and exposes the implicit assumptions these solutions often make.

- The dissertation also makes several contributions to the field of machine learning. It introduces representation matching as an important application of multistrategy learning. It develops the XML Learner, a novel approach that exploits the hierarchical nature of XML data to achieve better classification accuracy than existing learning approaches. Finally, it significantly extends relaxation labeling to address the problem of learning to label interrelated instances.

### *1.7 Outline*

The next chapter describes the representation matching problems that we consider in this dissertation. It elaborates on the ideas outlined in Section 1.5.1. The following three chapters – Chapters

3-5 – describe LSD, COMAP, and GLUE, respectively. They elaborate on the ideas outlined in Sections 1.5.2-1.5.4. Chapter 6 reviews existing solutions and discuss how they relate to ours. Finally, Chapter 7 summarizes the dissertation and discusses directions for future research.

The dissertation is structured so that each chapter is relatively self-contained. The impatient reader can read Chapters 1, 2, and 6 to quickly understand the main ideas and their relation to existing works. The remaining chapters can be read subsequently, as time permits.

Parts of this dissertation have been published in conferences and journals. In particular, the LSD system (Chapter 3) is described in a SIGMOD-2001 paper [DDH01], and the GLUE system (Chapter 5) is described in a WWW-2002 paper [DMDH02]. The key ideas of a multi-strategy learning approach are described in a Machine Learning Journal paper [DDH03].