The VLDB Journal manuscript No.
(will be inserted by the editor)

Learning to Match Ontologies on the Semantic Web

AnHai Doan[1], Jayant Madhavan[2], Robin Dhamankar[1], Pedro Domingos[2], Alon Halevy[2]

[1] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
{anhai,dhamanka}@cs.uiuc.edu

[2] Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195, USA
{jayant,pedrod,alon}@cs.washington.edu

Received: date / Revised version: date

Abstract On the Semantic Web, data will inevitably come from many different ontologies, and information processing across ontologies is not possible without knowing the semantic mappings between them. Manually finding such mappings is tedious, error-prone, and clearly not possible at the Web scale. Hence, the development of tools to assist in the ontology mapping process is crucial to the success of the Semantic Web. We describe GLUE, a system that employs machine learning techniques to find such mappings. Given two ontologies, for each concept in one ontology GLUE finds the most similar concept in the other ontology. We give well-founded probabilistic definitions to several practical similarity measures, and show that GLUE can work with all of them. Another key feature of GLUE is that it uses multiple learning strategies, each of which exploits well a different type of information either in the data instances or in the taxonomic structure of the ontologies. To further improve matching accuracy, we extend GLUE to incorporate commonsense knowledge and domain constraints into the matching process. Our approach is thus distinguished in that it works with a variety of well-defined similarity notions and that it efficiently incorporates multiple types of knowledge. We describe a set of experiments on several real-world domains, and show that GLUE proposes highly accurate semantic mappings. Finally, we extend GLUE to find complex mappings between ontologies, and describe experiments that show the promise of the approach.

Key words Semantic Web, Ontology Matching, Machine Learning, Relaxation Labeling.

1 Introduction

The current World-Wide Web has well over 1.5 billion pages [goo], but the vast majority of them are in human-readable format only (e.g., HTML). As a consequence software agents (softbots) cannot understand and process this information, and much of the potential of the Web has so far remained untapped.

In response, researchers have created the vision of the *Semantic Web* [BLHL01], where data has structure and *ontologies* describe the semantics of the data. When data is marked up using ontologies, softbots can better understand the semantics and therefore more intelligently locate and integrate data for a wide variety of tasks. The following example illustrates the vision of the Semantic Web.

Example 1 Suppose you want to find out more about someone you met at a conference. You know that his last name is Cook, and that he teaches Computer Science at a nearby university, but you do not know which one. You also know that he just moved to the US from Australia, where he had been an associate professor at his alma mater.

On the World-Wide Web of today you will have trouble finding this person. The above information is not contained within a single Web page, thus making keyword search ineffective. On the Semantic Web, however, you should be able to quickly find the answers. A marked-up directory service makes it easy for your personal softbot to find nearby Computer Science departments. These departments have marked up data using some ontology such as the one in Figure 1.a. Here the data is organized into a *taxonomy* that includes courses, people, and professors. Professors have *attributes* such as name, degree, and degree-granting institution (i.e., the one from which a professor obtained his or her Ph.D. degree). Such marked-up data makes it easy for your softbot to find a professor with the last name Cook. Then by examining the attribute "granting institution", the softbot quickly finds the alma mater CS department in Australia. Here, the softbot learns that the data has been marked up using an ontology specific to Australian universities, such as the one in Figure 1.b, and that there are many entities named Cook. However, knowing that "associate professor" is equivalent to "senior lecturer", the bot can select the right subtree in the departmental taxonomy, and zoom in on the old homepage of your conference acquaintance. □

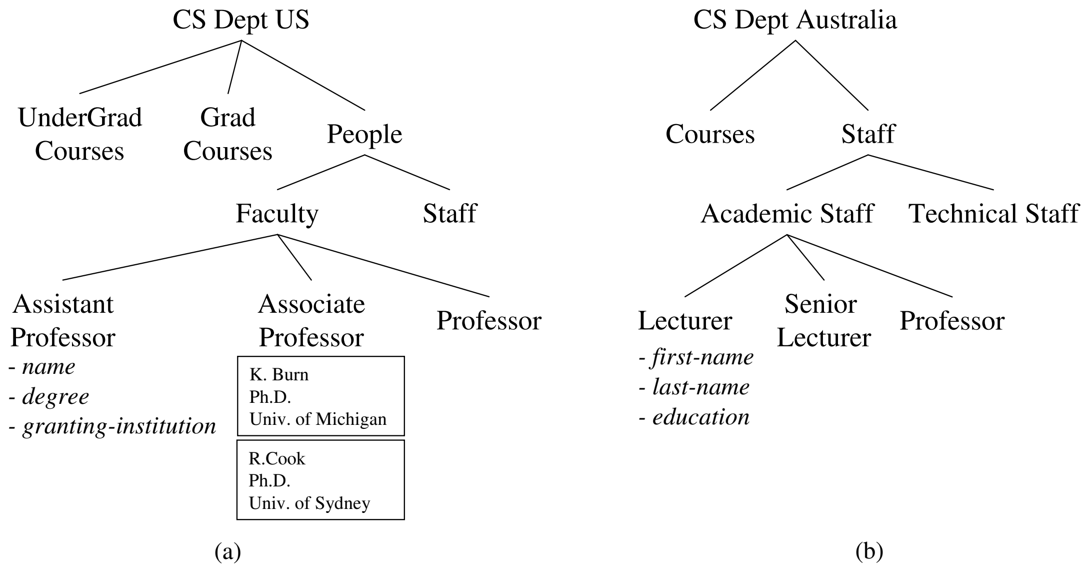

Fig. 1 Computer Science Department Ontologies.

The Semantic Web thus offers a compelling vision, but it also raises many difficult challenges. Researchers have been actively working on these challenges, focusing on fleshing out the basic architecture, developing expressive and efficient ontology languages, building techniques for efficient marking up of data, and learning ontologies (e.g., [HH01, BKD+01, Ome01, MS01, iee01]).

A key challenge in building the Semantic Web, one that has received relatively little attention, is finding *semantic mappings among the ontologies*. Given the de-centralized nature of the development of the Semantic Web, there will be an explosion in the number of ontologies. Many of these ontologies will describe similar domains, but using different terminologies, and others will have overlapping domains. To integrate data from disparate ontologies, we must know the *semantic correspondences* between their elements [BLHL01, Usc01]. For example, in the conference-acquaintance scenario described earlier, in order to find the right person, your softbot must know that "associate professor" in the US corresponds to "senior lecturer" in Australia. Thus, the semantic correspondences are in effect the "glue" that hold the ontologies together into a "web of semantics". Without them, the Semantic Web is akin to an electronic version of the Tower of Babel. Unfortunately, manually specifying such correspondences is time-consuming, error-prone [NM00], and clearly not possible on the Web scale. Hence, the development of tools to assist in ontology mapping is crucial to the success of the Semantic Web [Usc01].

2 Overview of Our Solution

In response to the challenge of ontology matching on the Semantic Web, we have developed the GLUE system, which applies machine learning techniques to semi-automatically create semantic mappings. Since taxonomies are central components of ontologies, we focus first on finding one-to-one (1-1) correspondences between the taxonomies of two given ontologies: for each concept node in one taxonomy, find the *most similar* concept node in the other taxonomy.

Similarity Definition: The first issue we address is the meaning of similarity between two concepts. Clearly, many different definitions of similarity are possible, each being appropriate for certain situations. Our approach is based on the observation that many practical measures of similarity can be defined based solely on the *joint probability distribution* of the concepts involved. Hence, instead of committing to a particular definition of similarity, GLUE calculates the joint distribution of the concepts, and lets the application use the joint distribution to compute any suitable similarity measure.

Specifically, for any two concepts A and B, the joint distribution consists of $P(A,B)$, $P(A,\overline{B})$, $P(\overline{A},B)$, and $P(\overline{A},\overline{B})$, where a term such as $P(A,\overline{B})$ is the probability that an instance in the domain belongs to concept A but not to concept B. An application can then define similarity to be a suitable function of these four values. For example, a similarity measure we use in this paper is $P(A \cap B)/P(A \cup B)$, otherwise known as the *Jaccard* coefficient [vR79].

Computing Similarities: The second challenge we address is that of computing the joint distribution of any two given concepts A and B. Under certain general assumptions (discussed in Section 5), a term such as $P(A,B)$ can be approximated as the fraction of data instances (in the data associated with the taxonomies or, more generally, in the probability distribution that generated the data) that belong to both A and B. Hence, the problem reduces to deciding for each data instance if it belongs to $A \cap B$. However, the input to our problem includes instances of A and instances of B in isolation. GLUE addresses this problem using machine learning techniques as follows: it uses the instances of A to learn a classifier for A, and then classifies instances of B according to that classifier, and vice-versa. Hence, we have a method for identifying instances of $A \cap B$.

Multi-Strategy Learning: Applying machine learning to our context raises the question of which learning algorithm to use and which types of information to exploit. Many different types of information can contribute toward the classification of an instance: its name, value format, the word frequencies in its value, and each of these is best utilized by a different learning algorithm. GLUE uses a *multi-strategy learning* approach [DDH01]: we employ a set of learners, then combine their predictions using a meta-learner. In previous work [DDH01] we have shown that multi-strategy learning is effective in the context of mapping between database schemas.

Exploiting Domain Constraints: GLUE also attempts to exploit available domain constraints and general heuristics in order to improve matching accuracy. An example heuristic is the observation that two nodes are likely to match if nodes in their neighborhood also match. An example of a domain constraint is "if node X matches Professor and node Y is an ancestor of X in the taxonomy, then it is unlikely that Y matches Assistant-Professor". Such constraints occur frequently in practice, and heuristics are commonly used when manually mapping between ontologies.

Previous works have exploited only one form or the other of such knowledge and constraints, in restrictive settings [NM01, MZ98, MBR01, MMGR02]. Here, we develop a unifying approach to incorporate all such types of information. Our approach is based on *relaxation labeling*, a powerful technique used extensively in the vision and image processing community [HZ83], and successfully adapted to solve matching and classification problems in 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 and domain constraints.

Handling Complex Mappings: Finally, we extend GLUE to build CGLUE, a system that finds complex mappings between two given taxonomies, such as "Courses maps to the union of Undergrad-Courses and Grad-Courses". CGLUE adapts the beam search technique commonly used in AI to efficiently discover such mappings.

Contributions: Our paper therefore makes the following contributions:

- We describe well-founded notions of semantic similarity, based on the joint probability distribution of the concepts involved. Such notions make our approach applicable to a broad range of ontology-matching problems that employ different similarity measures.
- We describe the use of multi-strategy learning for finding the joint distribution, and thus the similarity value of any concept pair in two given taxonomies. The GLUE system, embodying our approach, utilizes many different types of information to maximize matching accuracy. Multi-strategy learning also makes our system easily extensible to additional learners, as they become available.
- We introduce *relaxation labeling* to the ontology-matching context, and show that it can be adapted to efficiently exploit a broad range of common knowledge and domain constraints to further improve matching accuracy.
- We show that the GLUE approach can be extended to find complex mappings. The solution, as embodied by the CGLUE system, adapts beam search techniques to efficiently discover the mappings.
- We describe a set of experiments on several real-world domains to validate the effectiveness of GLUE and CGLUE. The results show the utility of multi-strategy learning and relaxation labeling, and that GLUE can work well with different notions of similarity. The results also show the promise of the CGLUE approach to finding complex mappings.

We envision the GLUE system to be a significant piece of a more complete ontology matching solution. We believe any such solution should have a significant user interaction component. Semantic mappings can often be highly subjective and depend on the choice of target application. User interaction is invaluable and indispensable in such cases. We do not address this in our current solution. However, the automated support that GLUE will provide to a more complete tool will significantly reduce the effort required of the user, and in many cases will reduce it to just mapping validation rather than construction.

Parts of the materials in this paper have appeared in [DMDH02, DMDH03, Doa02]. In those works we describe the problem of 1-1 matching for ontologies and the GLUE solution. In this paper, beyond a comprehensive description of GLUE, we also discuss the problem of finding complex mappings for ontologies and present a solution in form of the CGLUE system.

In the next section we define the ontology-matching problem. Section 4 discusses our approach to measuring similarity, and Sections 5-6 describe the GLUE system. Section 7 presents our experiments with GLUE. Section 8 extends GLUE to build CGLUE, then describes experiments with the system. Section 9 reviews related work. Section 10 discusses future work and concludes.

3 The Ontology Matching Problem

We now introduce ontologies, then define the problem of ontology matching. An *ontology* specifies a conceptualization of a domain in terms of concepts, attributes, and relations [Fen01]. The *concepts* provided model entities of interest in the domain. They are typically organized into a *taxonomy tree* where each node represents a concept and each concept is a specialization of its parent. Figure 1 shows two sample taxonomies for the CS department domain (which are simplifications of real ones).

Each concept in a taxonomy is associated with a set of *instances*. For example, concept Associate-Professor has instances "Prof. Cook" and "Prof. Burn" as shown in Figure 1.a. By the taxonomy's definition, the instances of a concept are also instances of an ancestor concept. For example, instances of Assistant-Professor, Associate-Professor, and Professor in Figure 1.a are also instances of Faculty and People.

Each concept is also associated with a set of *attributes*. For example, the concept Associate-Professor in Figure 1.a has the attributes name, degree, and granting-institution. An *instance* that belongs to a concept has fixed attribute values. For example, the instance "Professor Cook" has value name = "R. Cook", degree = "Ph.D.", and so on. An ontology also defines a set of *relations* among its concepts. For example, a relation AdvisedBy(Student,Professor) might list all instance pairs of Student and Professor such that the former is advised by the latter.

Many formal languages to specify ontologies have been proposed for the Semantic Web, such as OIL, DAML+OIL, OWL, SHOE, and RDF [owl, BKD$^+$01, dam, HH01, BG00]. Though these languages differ in their terminologies and expressiveness, the ontologies that they model essentially share the same features we described above.

Given two ontologies, the *ontology-matching* problem is to find semantic mappings between them. The simplest type of mapping is a *one-to-one (1-1)* mapping between the elements, such as "Associate-Professor to Senior-Lecturer", and "degree maps to education". Notice that mappings between different types of elements are possible, such as "the relation AdvisedBy(Student,Professor) maps to the attribute advisor of the concept Student". Examples of more complex types of mapping include "name maps to the concatenation of first-name and last-name", and "the union of Undergrad-Courses and Grad-Courses maps to Courses". In general, a mapping may be specified as a query that transforms instances in one ontology into instances in the other [CGL01].

In this paper we focus on finding mappings between the taxonomies. This is because taxonomies are central components of ontologies, and successfully matching them would greatly aid in matching the rest of the ontologies. Extending matching to attributes and relations is the subject of ongoing research.

We will begin by considering 1-1 matching for taxonomies. The specific problem that we consider is as follows: *given two taxonomies and their associated data instances, for each node (i.e., concept) in one taxonomy, find the most similar node in the other taxonomy, for a pre-defined similarity measure*. This is a very general problem setting that makes our approach applicable to a broad range of common ontology-related problems, such as ontology integration and data translation among the ontologies. Later, in Section 8 we will consider extending our solution for 1-1 matching to address the problem of complex matching between taxonomies.

Data instances: GLUE makes heavy use of the fact that we have data instances associated with the ontologies we are matching. We note that many real-world ontologies already have associated data instances. Furthermore, on the Semantic Web, the largest benefits of ontology matching come from matching the most heavily used ontologies; and the more heavily an ontology is used for marking up data, the more data it has. Finally, we show in our experiments that only a moderate number of data instances is necessary in order to obtain good matching accuracy.

4 Similarity Measures

To match concepts between two taxonomies, we need a notion of similarity. We now describe the similarity measures that GLUE handles; but before doing that, we discuss the motivations leading to our choices.

First, we would like the similarity measures to be well-defined. A well-defined measure will facilitate the evaluation of our system. It also makes clear to the users what the system means by a match, and helps them figure out whether the system is applicable to a given matching scenario. Furthermore, a well-defined similarity notion may allow us to leverage special-purpose techniques for the matching process.

Second, we want the similarity measures to correspond to our intuitive notions of similarity. In particular, they should depend only on the semantic content of the concepts involved, and not on their syntactic specification.

Finally, we note that many reasonable similarity measures exist, each being appropriate to certain situations. Hence, to maximize our system's applicability, we would like it to be able to handle a broad variety of similarity measures. The following examples illustrate the variety of possible definitions of similarity.

Example 2 In searching for your conference acquaintance, your softbot should use an "exact" similarity measure that maps Associate-Professor into Senior Lecturer, an equivalent concept. However, if the softbot has some postprocessing capabilities that allow it to filter data, then it may tolerate a "most-specific-parent" similarity measure that maps Associate-Professor to Academic-Staff, a more general concept. □

Example 3 A common task in ontology integration is to place a concept A into an appropriate place in a taxonomy T. One way to do this is to (a) use an "exact" similarity measure to find the concept B in T that is "most similar" to A, (b) use a "most-specific-parent" similarity measure to find the concept C in T that is the most specific superset concept of A, (c) use a "most-general-child" similarity measure to find the concept D in T that is the most general subset concept of A, then (d) decide on the placement of A, based on B, C, and D. □

Example 4 Certain applications may even have *different* similarity measures for different concepts. Suppose that a user tells the softbot to find houses in the range of \$300-500K, located in Seattle. The user expects that the softbot will not return houses that fail to satisfy the above criteria. Hence, the softbot should use exact mappings for price and address. But it may use approximate mappings for other concepts. If it maps house-description into neighborhood-info, that is still acceptable. □

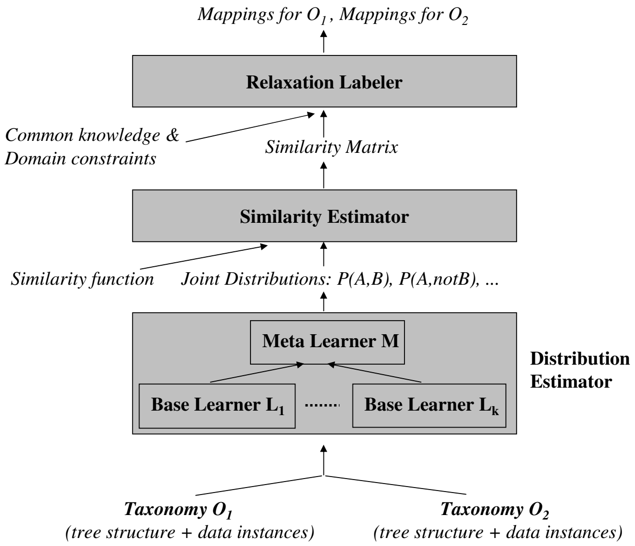

Fig. 2 The GLUE Architecture.

Most existing works in ontology (and schema) matching do not satisfy the above motivating criteria. Many works implicitly assume the existence of a similarity measure, but never define it. Others define similarity measures based on the syntactic clues of the concepts involved. For example, the similarity of two concepts might be computed as the dot product of the two TF/IDF (Term Frequency/Inverse Document Frequency) vectors representing the concepts, or a function based on the common tokens in the names of the concepts. Such similarity measures are problematic because they depend not only on the concepts involved, but also on their syntactic specifications.

4.1 Distribution-based Similarity Measures

We now give precise similarity definitions and show how our approach satisfies the motivating criteria. We begin by modeling each concept as a *set of instances*, taken from a *finite universe of instances*. In the CS domain, for example, the universe consists of all entities of interest in that world: professors, assistant professors, students, courses, and so on. The concept Professor is then the set of all instances in the universe that are professors. Given this model, the notion of the *joint probability distribution* between any two concepts A and B is well defined. This distribution consists of the four probabilities: $P(A,B), P(A,\overline{B}), P(\overline{A},B)$, and $P(\overline{A},\overline{B})$. A term such as $P(A,\overline{B})$ is the probability that a randomly chosen instance from the universe belongs to A but not to B, and is computed as the fraction of the universe that belongs to A but not to B.

Many practical similarity measures can be defined based on the joint distribution of the concepts involved. For instance, a possible definition for the "exact" similarity measure mentioned in the previous section is

$$\textit{Jaccard-sim}(A,B) = P(A \cap B)/P(A \cup B)$$

$$= \frac{P(A,B)}{P(A,B)+P(A,\overline{B})+P(\overline{A},B)} \quad (1)$$

This similarity measure is known as the *Jaccard* coefficient [vR79]. It takes the lowest value 0 when A and B are disjoint, and the highest value 1 when A and B are the same concept. Most of our experiments will use this similarity measure.

A definition for the "most-specific-parent" similarity measure is

$$MSP(A,B) = \begin{cases} P(A|B) & if\ P(B|A) = 1 \\ 0 & otherwise \end{cases} \quad (2)$$

where the probabilities $P(A|B)$ and $P(B|A)$ can be trivially expressed in terms of the four joint probabilities. This definition states that if B subsumes A, then the more specific B is, the higher $P(A|B)$, and thus the higher the similarity value $MSP(A,B)$ is. Thus it suits the intuition that the most specific parent of A in the taxonomy is the smallest set that subsumes A. An analogous definition can be formulated for the "most-general-child" similarity measure.

Instead of trying to estimate specific similarity values directly, GLUE focuses on computing the joint distributions. Then, it is possible to compute any of the above mentioned similarity measures as a function over the joint distributions. Hence, GLUE has the significant advantage of being able to work with a variety of similarity functions that have well-founded probabilistic interpretations.

5 The GLUE Architecture

We now describe GLUE in detail. The basic architecture of GLUE is shown in Figure 2. It consists of three main modules: *Distribution Estimator*, *Similarity Estimator*, and *Relaxation Labeler*.

The *Distribution Estimator* takes as input two taxonomies O_1 and O_2, together with their data instances. Then it applies machine learning techniques to compute for every pair of concepts $\langle A \in O_1, B \in O_2 \rangle$ their joint probability distribution. Recall from Section 4 that this joint distribution consists of four numbers: $P(A,B), P(A,\overline{B}), P(\overline{A},B)$, and $P(\overline{A},\overline{B})$. Thus a total of $4|O_1||O_2|$ numbers will be computed, where $|O_i|$ is the number of nodes (i.e., concepts) in taxonomy O_i. The *Distribution Estimator* uses a set of base learners and a meta-learner. We describe the learners and the motivation behind them in Section 5.2.

Next, GLUE feeds the above numbers into the *Similarity Estimator*, which applies a user-supplied similarity function (such as the ones in Equations 1 or 2) to compute a similarity value for each pair of concepts $\langle A \in O_1, B \in O_2 \rangle$. The output from this module is a *similarity matrix* between the concepts in the two taxonomies.

The *Relaxation Labeler* module then takes the similarity matrix, together with domain-specific constraints and heuristic knowledge, and searches for the mapping configuration that best satisfies the domain constraints and the common knowledge, taking into account the observed similarities. This mapping configuration is the output of GLUE.

We now describe the *Distribution Estimator*. First, we discuss the general machine-learning technique used to estimate joint distributions from data, and then the use of multi-strategy learning in GLUE. Section 6 describes the *Relaxation Labeler*. The *Similarity Estimator* is trivial because it simply applies a user-defined function to compute the similarity of two concepts from their joint distribution, and hence is not discussed further.

5.1 The Distribution Estimator

Consider computing the value of $P(A,B)$. This joint probability can be computed as the fraction of the instance universe that belongs to both A and B. In general we cannot compute this fraction because we do not know every instance in the universe. Hence, we must estimate $P(A,B)$ based on the data we have, namely, the instances of the two input taxonomies. Note that the instances that we have for the taxonomies may be overlapping, but are not necessarily so.

To estimate $P(A,B)$, we make the general assumption that the set of instances of each input taxonomy is a *representative sample* of the instance universe covered by the taxonomy. We denote by U_i the set of instances given for taxonomy O_i, by $N(U_i)$ the size of U_i, and by $N(U_i^{A,B})$ the number of instances in U_i that belong to both A and B.

With the above assumption, $P(A,B)$ can be estimated by the following equation:[1]

$$P(A,B) = [N(U_1^{A,B}) + N(U_2^{A,B})] \;/\; [N(U_1) + N(U_2)], \tag{3}$$

Computing $P(A,B)$ then reduces to computing $N(U_1^{A,B})$ and $N(U_2^{A,B})$. Consider $N(U_2^{A,B})$. We can compute this quantity if we know for each instance s in U_2 whether it belongs to both A and B. One part is easy: we already know whether s belongs to B – if it is explicitly specified as an instance of B or of any descendant node of B. Hence, we only need to decide whether s belongs to A.

This is where we use machine learning. Specifically, we partition U_1, the set of instances of ontology O_1, into the set of instances that belong to A and the set of instances that do not belong to A. Then, we use these two sets as positive and negative examples, respectively, to train a classifier for A. Finally, we use the classifier to predict whether instance s belongs to A.

It is often the case that the classifier returns not a simple "yes" or "no" answer, but rather a confidence score α in the range [0,1] for the "yes" answer. The score reflects the uncertainty of the classification. In such cases the score for the "no" answer can be computed as $1-\alpha$. Thus we regard the classification as "yes" if $\alpha \geq 1-\alpha$, and as "no" otherwise.

[1] Notice that $N(U_i^{A,B})/N(U_i)$ is also a reasonable approximation of $P(A,B)$, but it is estimated based only on the data of O_i. The estimation in (3) is likely to be more accurate because it is based on more data, namely, the data of both O_1 and O_2. Note also that the estimation in (3) is only an approximate in that it does not take into account the overlapping instances of the taxonomies.

In summary, we estimate the joint probability distribution of A and B as follows (the procedure is illustrated in Figure 3):

1. Partition U_1, into U_1^A and $U_1^{\overline{A}}$, the set of instances that do and do not belong to A, respectively (Figures 3.a-b).
2. Train a learner L for instances of A, using U_1^A and $U_1^{\overline{A}}$ as the sets of positive and negative training examples, respectively.
3. Partition U_2, the set of instances of taxonomy O_2, into U_2^B and $U_2^{\overline{B}}$, the set of instances that do and do not belong to B, respectively (Figures 3.d-e).
4. Apply learner L to each instance in U_2^B (Figure 3.e). This partitions U_2^B into the two sets $U_2^{A,B}$ and $U_2^{\overline{A},B}$ shown in Figure 3.f. Similarly, applying L to $U_2^{\overline{B}}$ results in the two sets $U_2^{A,\overline{B}}$ and $U_2^{\overline{A},\overline{B}}$.
5. Repeat Steps 1-4, but with the roles of taxonomies O_1 and O_2 being reversed, to obtain the sets $U_1^{A,B}$, $U_1^{\overline{A},B}$, $U_1^{A,\overline{B}}$, and $U_1^{\overline{A},\overline{B}}$.
6. Finally, compute $P(A,B)$ using Formula 3. The remaining three joint probabilities are computed in a similar manner, using the sets $U_2^{\overline{A},B}, \ldots, U_1^{\overline{A},\overline{B}}$ computed in Steps 4-5.

By applying the above procedure to all pairs of concepts $\langle A \in O_1, B \in O_2 \rangle$ we obtain all joint distributions of interest.

5.2 Multi-Strategy Learning

Given the diversity of machine learning methods, the next issue is deciding which one to use for the procedure we described above. A key observation in our approach is that there are *many* different types of information that a learner can glean from the training instances, in order to make predictions. It can exploit the *frequencies* of words in the text value of the instances, the instance *names*, the value *formats*, the *characteristics of value distributions*, and so on.

Since different learners are better at utilizing different types of information, GLUE follows [DDH01] and takes a *multi-strategy learning* approach. In Step 2 of the above estimation procedure, instead of training a single learner L, we train a set of learners $L_1, \ldots, L_k$, called *base learners*. Each base learner exploits well a certain type of information from the training instances to build prediction hypotheses. Then, to classify an instance in Step 4, we apply the base learners to the instance and combine their predictions using a *meta-learner*. This way, we can achieve higher classification accuracy than with any single base learner alone, and therefore better approximations of the joint distributions.

The current implementation of GLUE has two base learners, *Content Learner* and *Name Learner*, and a meta-learner that is a linear combination of the base learners. We now describe these learners in detail.

The Content Learner: This learner exploits the frequencies of words in the *textual content* of an instance to make predictions. Recall that an instance typically has a *name* and a set of

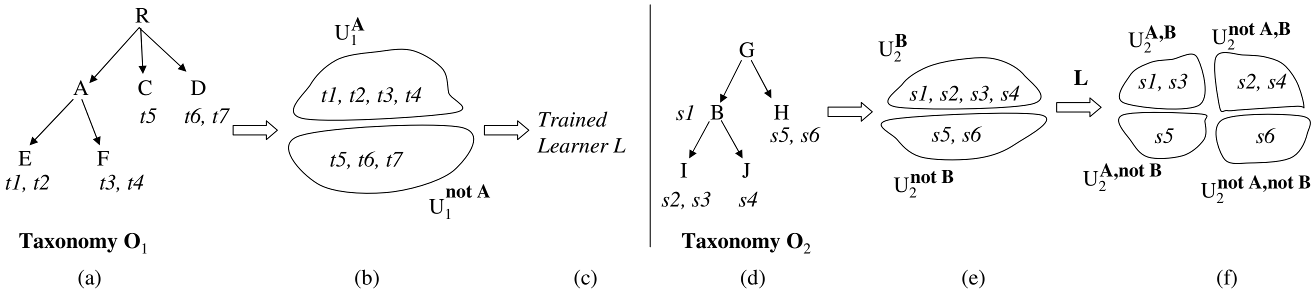

Fig. 3 Estimating the joint distribution of concepts A and B.

attributes together with their values. In the current version of GLUE, we do not handle attributes directly; rather, we treat them and their values as the *textual content* of the instance[2]. For example, the textual content of the instance "Professor Cook" is "R. Cook, Ph.D., University of Sidney, Australia". The textual content of the instance "CSE 342" is the text content of this course' homepage.

The Content Learner employs the Naive Bayes learning technique [DP97], one of the most popular and effective text classification methods. It treats the textual content of each input instance as a *bag of tokens*, which is generated by parsing and stemming the words and symbols in the content. Let $d = \{w_1, \ldots, w_k\}$ be the content of an input instance, where the w_j are tokens. To make a prediction, the Content Learner needs to compute the probability that an input instance is an instance of A, given its tokens, i.e., $P(A|d)$.

Using Bayes' theorem, $P(A|d)$ can be rewritten as $P(d|A)P(A)/P(d)$. Fortunately, two of these values can be estimated using the training instances, and the third, $P(d)$, can be ignored because it is just a normalizing constant. Specifically, $P(A)$ is estimated as the portion of training instances that belong to A. To compute $P(d|A)$, we assume that the tokens w_j appear in d *independently* of each other given A (this is why the method is called *naive* Bayes). With this assumption, we have

$$P(d|A) = P(w_1|A)P(w_2|A)\cdots P(w_k|A)$$

$P(w_j|A)$ is estimated as $n(w_j, A)/n(A)$, where $n(A)$ is the total number of token positions of all training instances that belong to A, and $n(w_j, A)$ is the number of times token w_j appears in all training instances belonging to A. Even though the independence assumption is typically not valid, the Naive Bayes learner still performs surprisingly well in many domains, notably text-based ones (see [DP97] for an explanation).

We compute $P(\overline{A}|d)$ in a similar manner. Hence, the Content Learner predicts A with probability $P(A|d)$, and $\overline{A}$ with the probability $P(\overline{A}|d)$.

The Content Learner works well on long textual elements, such as course descriptions, or elements with very distinct and descriptive values, such as color (red, blue, green, etc.). It is less effective with short, numeric elements such as course numbers or credits.

The Name Learner: This learner is similar to the Content Learner, but makes predictions using the *full name* of the input instance, instead of its *content*. The full name of an instance is the concatenation of concept names leading from the root of the taxonomy to that instance. For example, the full name of instance with the name s_4 in taxonomy O_2 (Figure 3.d) is "G B J s_4". This learner works best on specific and descriptive names. It does not do well with names that are too vague or vacuous.

The Meta-Learner: The predictions of the base learners are combined using the meta-learner. The meta-learner assigns to each base learner a *learner weight* that indicates how much it *trusts* that learner's predictions. Then it combines the base learners' predictions via a weighted sum.

For example, suppose the weights of the Content Learner and the Name Learner are 0.6 and 0.4, respectively. Suppose further that for instance s_4 of taxonomy O_2 (Figure 3.d) the Content Learner predicts A with probability 0.8 and $\overline{A}$ with probability 0.2, and the Name Learner predicts A with probability 0.3 and $\overline{A}$ with probability 0.7. Then the Meta-Learner predicts A with probability $0.8 \cdot 0.6 + 0.3 \cdot 0.4 = 0.6$ and $\overline{A}$ with probability $0.2 \cdot 0.6 + 0.7 \cdot 0.4 = 0.4$.

In the current GLUE system, the learner weights are set manually, based on the characteristics of the base learners and the taxonomies. However, they can also be set automatically using a machine learning approach called *stacking* [Wol92, TW99], as we have shown in [DDH01].

6 Exploiting Domain Constraints and Heuristic Knowledge

We now describe the *Relaxation Labeler*, which takes the similarity matrix from the *Similarity Estimator*, and searches for the mapping configuration that best satisfies the given domain constraints and heuristic knowledge. We first describe relaxation labeling, then discuss the domain const- raints and heuristic knowledge employed in our approach.

[2] However, more sophisticated learners can be developed that deal explicitly with the attributes, such as the XML Learner in [DDH01].

6.1 Relaxation Labeling

Relaxation labeling is an efficient technique to solve the problem of assigning labels to nodes of a graph, given a set of constraints. The key idea behind this approach is that the label of a node is typically influenced by the *features of the node's neighborhood* in the graph. Examples of such features are the labels of the neighboring nodes, the percentage of nodes in the neighborhood that satisfy a certain criterion, and the fact that a certain constraint is satisfied or not.

Relaxation labeling exploits this observation. The influence of a node's neighborhood on its label is quantified using a formula for the probability of each label as a function of the neighborhood features. Relaxation labeling assigns initial labels to nodes based solely on the intrinsic properties of the nodes. Then it performs *iterative local optimization*. In each iteration it uses the formula to change the label of a node based on the features of its neighborhood. This continues until labels do not change from one iteration to the next, or some other convergence criterion is reached.

Relaxation labeling appears promising for our purposes because it has been applied successfully to similar matching problems in computer vision, natural language processing, and hypertext classification [HZ83, Pad98, CDI98]. It is relatively efficient, and can handle a broad range of constraints. Even though its convergence properties are not yet well understood (except in certain cases) and it is liable to converge to a local maxima, in practice it has been found to perform quite well [Pad98, CDI98].

We now explain how to apply relaxation labeling to the problem of mapping from taxonomy O_1 to taxonomy O_2. We regard nodes (concepts) in O_2 as *labels*, and recast the problem as finding the best label assignment to nodes (concepts) in O_1, given all knowledge we have about the domain and the two taxonomies.

Our goal is to derive a formula for updating the probability that a node takes a label based on the features of the neighborhood. Let X be a node in taxonomy O_1, and L be a label (i.e., a node in O_2). Let Δ_K represent all that we know about the domain, namely, the tree structures of the two taxonomies, the sets of instances, and the set of domain constraints. Then we have the following conditional probability

$$\begin{aligned} P(X = L|\Delta_K) &= \sum_{M_X} P(X = L, M_X|\Delta_K) \\ &= \sum_{M_X} P(X = L|M_X, \Delta_K)P(M_X|\Delta_K) \end{aligned} \quad (4)$$

where the sum is over all possible label assignments M_X to all nodes other than X in taxonomy O_1. Assuming that the nodes' label assignments are independent of each other given Δ_K, we have

$$P(M_X|\Delta_K) = \prod_{(X_i=L_i)\in M_X} P(X_i = L_i|\Delta_K) \quad (5)$$

Consider $P(X = L|M_X, \Delta_K)$. M_X and Δ_K constitutes all that we know about the neighborhood of X. Suppose now

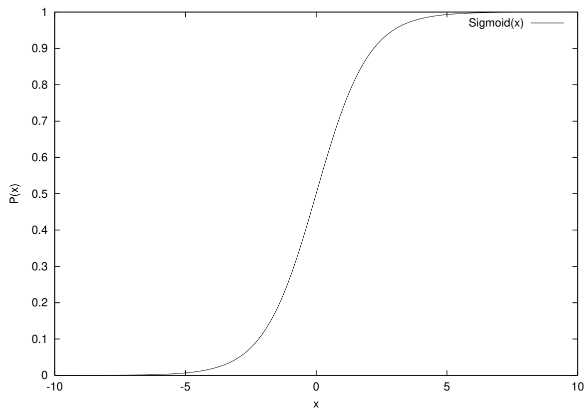

Fig. 4 The sigmoid function

that the probability of X getting label L depends only on the values of n features of this neighborhood, where each feature is a function $f_i(M_X, \Delta_K, X, L)$. As we explain later in this section, each such feature corresponds to one of the heuristics or domain constraints that we wish to exploit. Then

$$P(X = L|M_X, \Delta_K) = P(X = L|f_1, \ldots, f_n) \quad (6)$$

If we have access to previously-computed mappings between taxonomies in the same domain, we can use them as the training data from which to estimate $P(X = L|f_1, \ldots, f_n)$ (see [CDI98] for an example of this in the context of hypertext classification). However, here we will assume that such mappings are not available. Hence we use alternative methods to quantify the influence of the features on the label assignment. In particular, we use the sigmoid or logistic function $\sigma(x) = 1/(1 + e^{-x})$, where x is a linear combination of the features f_k, to estimate the above probability. This function is widely used to combine multiple sources of evidence [Agr90]. The general shape of the sigmoid is as shown in Figure 4.

Thus:

$$P(X = L|f_1, \ldots, f_n) \propto \sigma(\alpha_1 \cdot f_1 + \cdots + \alpha_n \cdot f_n) \quad (7)$$

where $\propto$ denotes "proportional to", and the weight α_k indicates the importance of feature f_k.

The sigmoid is essentially a smoothed threshold function, which makes it a good candidate for use in combining evidence from the different features. If the total evidence is below a certain value, it is unlikely that the nodes match; above this threshold, they probably do.

By substituting Equations 5-7 into Equation 4, we obtain

$$P(X = L|\Delta_K) \propto \sum_{M_X} \sigma\left(\sum_{k=1}^{n} \alpha_k f_k(M_X, \Delta_K, X, L)\right) \times \prod_{(X_i=L_i)\in M_X} P(X_i = L_i|\Delta_K) \quad (8)$$

The proportionality constant is found by renormalizing the probabilities of all the labels to sum to one. Notice that this equation expresses the probabilities $P(X = L|\Delta_K)$ for the various nodes in terms of each other. This is the iterative equation that we use for relaxation labeling.

Table 1 Examples of constraints that can be exploited to improve matching accuracy.

Constraint Types		Examples
Domain-Independent	*Neighborhood*	Two nodes match if their children also match. Two nodes match if their parents match and at least x% of their children also match. Two nodes match if their parents match and some of their descendants also match.
	Union	If all children of node X match node Y, then X also matches Y.
Domain-Dependent	*Subsumption*	If node Y is a descendant of node X, and Y matches PROFESSOR, then it is unlikely that X matches ASSISTANT-PROFESSOR. If node Y is NOT a descendant of node X, and Y matches PROFESSOR, then it is unlikely that X matches FACULTY.
	Frequency	There can be at most one node that matches DEPARTMENT-CHAIR.
	Nearby	If a node in the neighborhood of node X matches ASSOCIATE-PROFESSOR, then the chance that X matches PROFESSOR isincreased.

6.2 Constraints

Table 1 shows examples of the constraints currently used in our approach and their characteristics. We distinguish between two types of constraints: domain-independent and -dependent constraints. *Domain-independent constraints* convey our general knowledge about the interaction between related nodes. Perhaps the most widely used such constraint is the *Neighborhood Constraint*: "two nodes match if nodes in their neighborhood also match", where the neighborhood is defined to be the children, the parents, or both [NM01,MBR01,MZ98] (see Table 1). Another example is the *Union Constraint*: "if all children of a node A match node B, then A also matches B". This constraint is specific to the taxonomy context. It exploits the fact that A is the union of all its children. *Domain-dependent constraints* convey our knowledge about the interaction between specific nodes in the taxonomies. Table 1 shows examples of three types of domain-dependent constraints.

To incorporate the constraints into the relaxation labeling process, we model each constraint c_i as a feature f_i of the neighborhood of node X. For example, consider the constraint c_1: "two nodes are likely to match if their children match". To model this constraint, we introduce the feature $f_1(M_X, \Delta_K, X, L)$ that is the percentage of X's children that match a child of L, under the given M_X mapping. Thus f_1 is a numeric feature that takes values from 0 to 1. Next, we assign to f_i a *positive* weight α_i. This has the intuitive effect that, all other things being equal, the higher the value f_i (i.e., the percentage of matching children), the higher the probability of X matching L is.

As another example, consider the constraint c_2: "if node Y is a descendant of node X, and Y matches PROFESSOR, then it is unlikely that X matches ASST-PROFESSOR". The corresponding feature, $f_2(M_X, \Delta_K, X, L)$, is 1 if the condition "there exists a descendant of X that matches PROFESSOR" is satisfied, given the M_X mapping configuration, and 0 otherwise. Clearly, when this feature takes value 1, we want to substantially reduce the probability that X matches ASST-PROFESSOR. We model this effect by assigning to f_2 a *negative* weight α_2.

6.3 Efficient Implementation of Relaxation Labeling

In this section we discuss why previous implementations of relaxation labeling are not efficient enough for ontology matching, then describe an efficient implementation for our context.

Recall from Section 6.1 that our goal is to compute for each node X and label L the probability $P(X = L|\delta_K)$, using Equation 8. A naive implementation of this computation process would enumerate *all* labeling configurations M_X, then compute $f_k(M_X, \delta_K, X, L)$ for each of the configurations.

This naive implementation does not work in our context because of the vast number of configurations. This is a problem that has also arisen in the context of relaxation labeling being applied to hypertext classification ([CDI98]). The solution in [CDI98] is to consider only the top k configurations, that is, those with highest probability, based on the heuristic that the sum of the probabilities of the top k configurations is already sufficiently close to 1. This heuristic was true in the context of hypertext classification, due to a relatively small number of neighbors per node (in the range 0-30) and a relatively small number of labels (under 100).

Unfortunately the above heuristic is not true in our matching context. Here, a neighborhood of a node can be the entire graph, thereby comprising hundreds of nodes, and the number of labels can be hundreds or thousands (because this number is the same as the number of nodes in the other ontology to be matched). Thus, the number of configurations in our context is orders of magnitude more than that in the context of hypertext classification, and the probability of a configuration is computed by multiplying the probabilities of a very large number of nodes. As a consequence, even the highest probability of a configuration is very small, and a huge number of configurations have to be considered to achieve a significant total probability mass.

Hence we developed a novel and efficient implementation for relaxation labeling in our context. Our implementation relies on three key ideas. The first idea is that we divide the space of configurations into *partitions* $C_1, C_2, \ldots, C_m$, such that all configurations that belong to the same partition have the same values for the features $f_1, f_2, \ldots, f_n$. Then, to compute $P(X = L|\delta_K)$, we iterate over the (far fewer) partitions rather than over the huge space of configurations.

The one problem remaining is to compute the probability of a partition C_i. Suppose all configurations in C_i have feature values $f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n$. Our second key idea is to approximate the probability of C_i with $\prod_{j=1}^{n} P(f_j = v_j)$, where $P(f_j = v_j)$ is the total probability of all configurations whose feature f_j takes on value v_j. Note that this approximation makes an independence assumption over the features, which is clearly not valid. However, the assumption greatly simplifies the computation process. In our experiments with GLUE, we have not observed any problem arising because of this assumption.

Now we focus on computing $P(f_j = v_j)$. We compute this probability using a variety of techniques that depend on the particular feature. For example, suppose f_j is the number of children of X that map to some child of L. Let X_j be the j^{th} child of X (ordered arbitrarily) and n_X be the number of children of the concept X. Let S_j^m be the probability that of the first j children, there are m that are mapped to some child of L. It is easy to see that S_j^m's are related as follows,

$$S_j^m = P(X_j = L')S_{j-1}^{m-1} + (1 - P(X_j = L'))S_{j-1}^m$$

where $P(X_j = L') = \sum_{l=1}^{n_L} P(X_j = L_l)$ is the probability that the child X_j is mapped to some child of L. This equation immediately suggests a dynamic programming approach to computing the values S_j^m and thus the number of children of X that map to some child of L. We use similar techniques to compute $P(f_j = v_j)$ for the other types of features that are described in Table 1.

7 Empirical Evaluation

We have evaluated GLUE on several real-world domains. Our goals were to evaluate the matching accuracy of GLUE, to measure the relative contribution of the different components of the system, and to verify that GLUE can work well with a variety of similarity measures.

Domains and Taxonomies: We evaluated GLUE on three domains, whose characteristics are shown in Table 2. The domains Course Catalog I and II describe courses at Cornell University and the University of Washington. The taxonomies of Course Catalog I have 34 - 39 nodes, and are fairly similar to each other. The taxonomies of Course Catalog II are much larger (166 - 176 nodes) and much less similar to each other. Courses are organized into schools and colleges, then into departments and centers within each college. The Company Profile domain uses ontologies from Yahoo.com and TheStandard.com and describes the current business status of companies. Companies are organized into sectors, then into industries within each sector[3].

[3] Many ontologies are also available from research resources (e.g., DAML.org, semanticweb.org, OntoBroker [ont], SHOE, OntoAgents). However, they currently have no or very few data instances.

In each domain we downloaded two taxonomies. For each taxonomy, we downloaded the entire set of data instances, and performed some trivial data cleaning such as removing HTML tags and phrases such as "course not offered" from the instances. We also removed instances of size less than 130 bytes, because they tend to be empty or vacuous, and thus do not contribute to the matching process. We then removed all nodes with fewer than 5 instances, because such nodes cannot be matched reliably due to lack of data.

Similarity Measure & Manual Mappings: We chose to evaluate GLUE using the *Jaccard* similarity measure (Section 4), because it corresponds well to our intuitive understanding of similarity. Given the similarity measure, we manually created the correct 1-1 mappings between the taxonomies in the same domain, for evaluation purposes. The rightmost column of Table 2 shows the number of manual mappings created for each taxonomy. For example, we created 236 one-to-one mappings from *Standard* to *Yahoo!*, and 104 mappings in the reverse direction. Note that in some cases there were nodes in a taxonomy for which we could not find a 1-1 match. This was either because there was no equivalent node (e.g., School of Hotel Administration at Cornell has no equivalent counterpart at the University of Washington), or when it is impossible to determine an accurate match without additional domain expertise.

Domain Constraints: We specified domain constraints for the relaxation labeler. For the taxonomies in Course Catalog I, we specified all applicable subsumption constraints (see Table 1). For the other two domains, because their sheer size makes specifying all constraints difficult, we specified only the most obvious subsumption constraints (about 10 constraints for each taxonomy). For the taxonomies in Company Profiles we also used several frequency constraints.

Experiments: For each domain, we performed two experiments. In each experiment, we applied GLUE to find the mappings from one taxonomy to the other. The *matching accuracy* of a taxonomy is then the percentage of the manual mappings (for that taxonomy) that GLUE predicted correctly.

7.1 Matching Accuracy

Figure 5 shows the matching accuracy for different domains and configurations of GLUE. In each domain, we show the matching accuracy of two scenarios: mapping from the first taxonomy to the second, and vice versa. The four bars in each scenario (from left to right) represent the accuracy produced by: (1) the name learner alone, (2) the content learner alone, (3) the meta-learner using the previous two learners, and (4) the relaxation labeler on top of the meta-learner (i.e., the complete GLUE system).

The results show that GLUE achieves high accuracy across all three domains, ranging from 66 to 97%. In contrast, the best matching results of the base learners, achieved by the content learner, are only 52 - 83%. It is interesting that the name learner achieves very low accuracy, 12 - 15% in four

Table 2 Domains and taxonomies for our experiments.

Taxonomies		# nodes	# non-leaf nodes	depth	# instances in taxonomy	max # instances at a leaf	max # children of a node	# manual mappings created
Course Catalog I	*Cornell*	34	6	4	1526	155	10	34
	Washington	39	8	4	1912	214	11	37
Course Catalog II	*Cornell*	176	27	4	4360	161	27	54
	Washington	166	25	4	6957	214	49	50
Company Profiles	*Standard.com*	333	30	3	13634	222	29	236
	Yahoo.com	115	13	3	9504	656	25	104

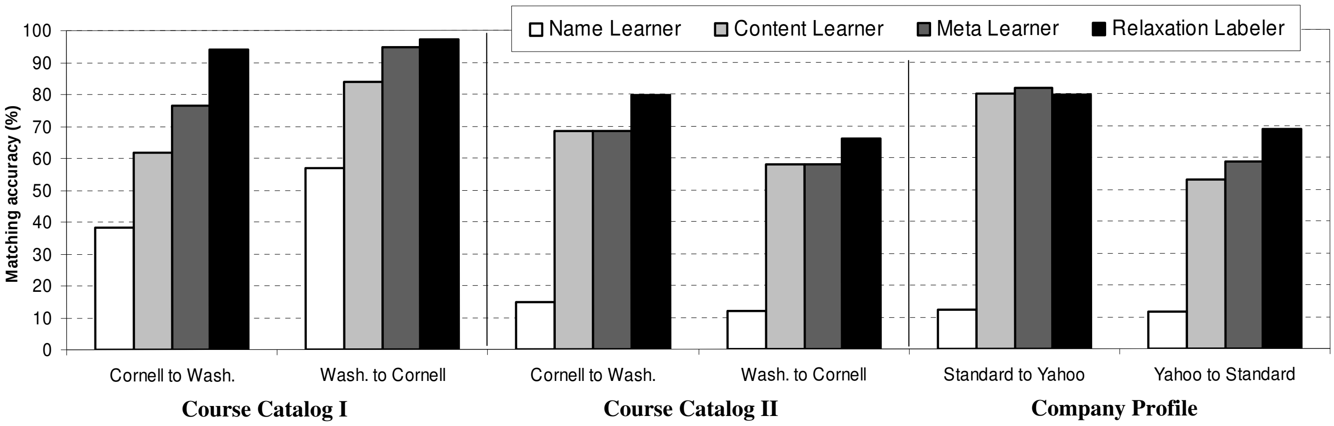

Fig. 5 Matching accuracy of GLUE.

out of six scenarios. This is because all instances of a concept, say B, have very similar full names (see the description of the name learner in Section 5.2). Hence, when the name learner for a concept A is applied to B, it will classify *all* instances of B as A or $\overline{A}$. In cases when this classfication is incorrect, which might be quite often, using the name learner alone leads to poor estimates of the joint distributions. The poor performance of the name learner underscores the importance of data instances and multi-strategy learning in ontology matching.

The results clearly show the utility of the meta-learner and relaxation labeler. Even though in half of the cases the meta-learner only minimally improves the accuracy, in the other half it makes substantial gains, between 6 and 15%. And in all but one case, the relaxation labeler further improves accuracy by 3 - 18%, confirming that it is able to exploit the domain constraints and general heuristics. In one case (from Standard to Yahoo), the relaxation labeler decreased accuracy by 2%. The performance of the relaxation labeler is discussed in more detail below. In Section 7.4 we identify the reasons that prevent GLUE from identifying the remaining mappings.

In the current experiments, GLUE utilized on average only 30 to 90 data instances per leaf node (see Table 2). The high accuracy in these experiments suggests that GLUE can work well with only a modest amount of data.

7.2 Performance of the Relaxation Labeler

In our experiments, when the relaxation labeler was applied, the accuracy typically improved substantially in the first few iterations, then gradually dropped. This phenomenon has also been observed in many previous works on relaxation labeling [HZ83, Llo83, Pad98]. Because of this, finding the right stopping criterion for relaxation labeling is of crucial importance. Many stopping criteria have been proposed, but no general effective criterion has been found.

We considered three stopping criteria: (1) stopping when the mappings in two consecutive iterations do not change (the *mapping criterion*), (2) when the probabilities do not change, or (3) when a fixed number of iterations has been reached.

We observed that when using the last two criteria the accuracy sometimes improved by as much as 10%, but most of the time it decreased. In contrast, when using the mapping criterion, in all but one of our experiments the accuracy substantially improved, by 3 - 18%, and hence, our results are reported using this criterion. We note that with the mapping criterion, we observed that relaxation labeling always stopped in the first few iterations.

In all of our experiments, relaxation labeling was also very fast. It took only a few seconds in Catalog I and under 20 seconds in the other two domains to finish ten iterations. This observation shows that relaxation labeling can be implemented efficiently in the ontology-matching context. It

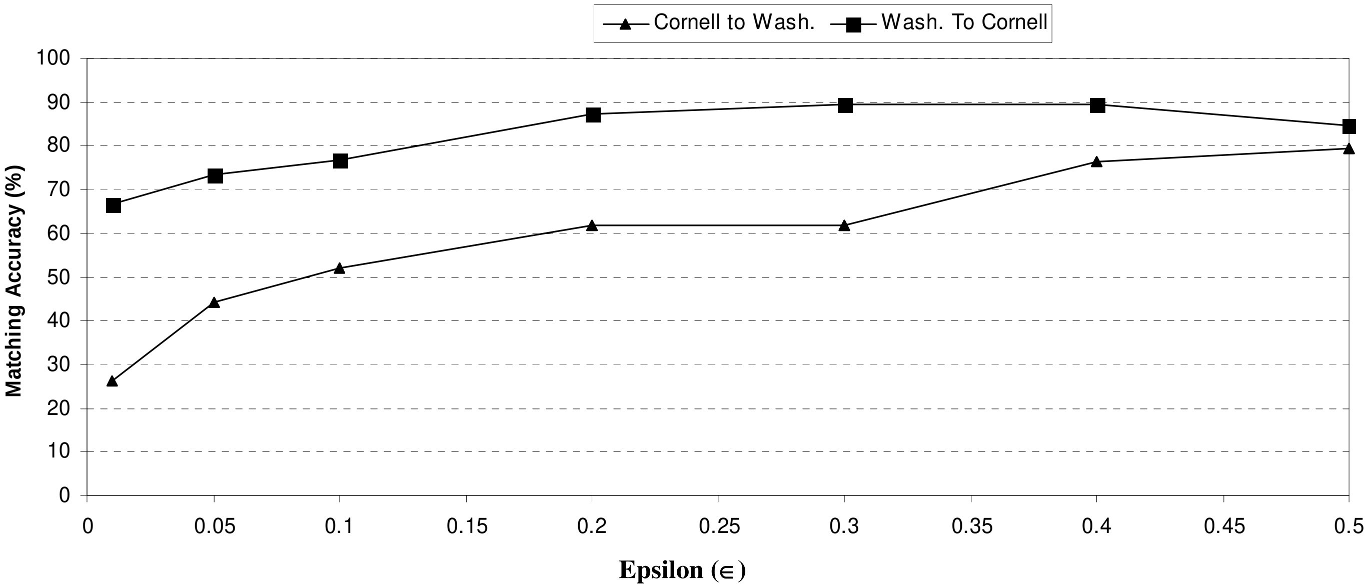

Fig. 6 The accuracy of GLUE in the Course Catalog I domain, using the most-specific-parent similarity measure.

also suggests that we can efficiently incorporate user feedback into the relaxation labeling process in the form of additional domain constraints.

We also experimented with different values for the constraint weights (see Section 6), and found that the relaxation labeler was quite robust with respect to such parameter changes.

7.3 Most-Specific-Parent Similarity Measure

So far we have experimented only with the *Jaccard* similarity measure. We wanted to know whether GLUE can work well with other similarity measures. Hence we conducted an experiment in which we used GLUE to find mappings for taxonomies in the Course Catalog I domain, using the following similarity measure:

$$MSP(A,B) = \begin{cases} P(A|B) & if\ P(B|A) \geq 1-\epsilon \\ 0 & otherwise \end{cases}$$

This measure is the same as the the *most-specific-parent* similarity measure described in Section 4, except that we added an ϵ factor to account for the error in approximating $P(B|A)$.

Figure 6 shows the matching accuracy, plotted against ϵ. As can be seen, GLUE performed quite well on a broad range of ϵ. This illustrates how GLUE can be effective with more than one similarity measure.

7.4 Discussion

The accuracy of GLUE is quite impressive as is, but it is natural to ask what limits GLUE from obtaining even higher accuracy. There are several reasons that prevent GLUE from correctly matching the remaining nodes. First, some nodes cannot be matched because of insufficient training data. For example, many course descriptions in Course Catalog II contain only vacuous phrases such as "3 credits". While there is clearly no general solution to this problem, in many cases it can be mitigated by adding base learners that can exploit domain characteristics to improve matching accuracy.

Second, the relaxation labeler performed local optimizations, and sometimes converged to only a local maxima, thereby not finding correct mappings for all nodes. Here, the challenge will be in developing search techniques that work better by taking a more "global perspective", but still retain the runtime efficiency of local optimization.

Third, the two base learners we used in our implementation are rather simple general-purpose text classifiers. Using other leaners that perform domain-specific feature selection and comparison can also improve the accuracy.

We note that some nodes cannot be matched automatically because they are simply ambiguous. For example, it is not clear whether "networking and communication devices" should match "communication equipment" or "computer networks". A solution to this problem is to incorporate user interaction into the matching process [NM00, DDH01, YMHF01].

Finally, GLUE currently tries to predict the best match for *every* node in the taxonomy. However, in some cases, such a match simply does not exist (e.g., unlike Cornell, the University of Washington does not have a School of Hotel Administration). Hence, an additional extension to GLUE is to make it be aware of such cases, and not predict an incorrect match when this occurs.

8 Extending GLUE to Complex Matching

GLUE finds 1-1 mappings between two given taxonomies. However, complex mappings are also widespread in practice. Hence, we extend GLUE to find such mappings. As earlier, we focus on *complex mappings between taxonomies*, such as

1. Let the initial set of candidates $\mathcal{C}$ be the set of all nodes of O_2. Set $highest\_sim = 0$.
2. Loop
   (a) Compute similarity score between each candidate of $\mathcal{C}$ and A.
   (b) Let $new\_highest\_sim$ be the highest similarity score of candidates of $\mathcal{C}$.
   (c) If $|new\_highest\_sim - highest\_sim| \leq \epsilon$, for a pre-specified ϵ, then stop, returning the candidate with the highest similarity score in $\mathcal{C}$.
   (d) Otherwise, select the k candidates with the highest score from $\mathcal{C}$. Expand these candidates to create new candidates. Add the new candidates to $\mathcal{C}$. Set $highest\_sim = new\_highest\_sim$.

Fig. 7 Finding the best mapping candidate for a node A of taxonomy O_1.

"Courses of the CS Dept Australia taxonomy maps to the union of Undergrad-Courses and Grad-Courses of the CS Dept US taxonomy" (Figure 1). Finding other types of complex mappings (e.g., "attribute name maps to the concatenation of first-name and last-name") is the subject of future research.

We consider the following specific matching problem: for each node A of a given taxonomy O_1, find the *best* mapping over the nodes of another taxonomy O_2 – be it a 1-1 or complex mapping. A 1-1 mapping has the form $A = X$ where X is a node of O_2. A complex mapping has the form $A = X_1\ op_1\ X_2\ op_2\ \ldots\ op_{n-1}\ X_n$, where the X_i are nodes of O_2 and the op_i are pre-defined operators. (In future work we shall consider many-to-many complex mappings such as $A_1\ op_1 A_2 = X_1\ op_2\ X_2\ op_3\ X_3$.) Since a taxonomic node is usually interpreted as a *set* of instances, we shall take the op_i to be set-theoretic operators: union, difference, complementary, etc.

In our matching context, we shall refer to a "composite concept" such as $X_1\ op_1\ X_2\ op_2\ \ldots\ op_{n-1}\ X_n$ as a *mapping candidate*. Since any set-arithmetic expression can be rewritten using only the union and difference operators, it follows that for any node A of O_1, we only need to consider mapping candidates that are built using these two operators.

Further, in the rest of this section we make the assumption that the children of any taxonomic node are mutually exclusive and exhaustive. That is, the children $C_1, C_2, \ldots, C_k$ of any node D (of O_1 or O_2) satisfy the conditions $C_i \cap C_j = \emptyset, 1 \leq i, j \leq k$ and $i \neq j$, and $C_1 \cup C_2 \cup \ldots \cup C_k = D$. In Section 8.4 we discuss removing this assumption, but here we note that the assumption holds for many real-world taxonomies, in which the further specialization of a node usually provides a partition of the instances of that node. In many other real-world taxonomies, such as the "course catalog" and "company profiles" domains we have considered in this paper, very few sibling nodes share instances, and the set of such instances is usually small. Thus, for these domains we can also make this approximating assumption.

With the above assumption, it is easy to show that any mapping candidate can be rewritten to be a *union* of nodes. Thus, for each node A of taxonomy O_1, our goal is to find the most similar mapping candidate from the set of candidates that are unions of nodes of taxonomy O_2.

8.1 The CGLUE System

To find the best mapping candidate for node A of taxonomy O_1, we can simply enumerate all "union" candidates over taxonomy O_2, compute for each candidate its similarity with respect to A, using the learning methods described in Section 5, then return the candidate with the highest similarity. However, since the number of candidates is exponential in terms of the number of nodes of O_2, the above brute-force approach is clearly impractical. Thus, we consider an *approximate* approach that casts the matching problem as that of *searching* through the huge space of candidates. To conduct an efficient search, we adapt the *beam search* technique commonly used in AI. The basic idea of beam search is that at each stage in the search process, we limit our attention to only k most promising candidates, where k is a pre-specified number.

The adapted beam search algorithm to find the best mapping candidate for a node A of O_1 is described in Figure 7. Here, in Step 2.a the algorithm computes the similarity score between a mapping candidate and node A using the learning method described in Section 5. This computation has been implemented on top of the current GLUE system. In Step 2.c, ϵ is currently set to be zero. In Step 2.d, for each candidate C in the set of selected k candidates, the algorithm unions C with nodes of O_2, thus generating $|O_2|$ potential new candidates. Next, it removes previously seen candidates as well as those that contain duplicate nodes. Since each candidate is just a union of nodes of O_2, the removal process could be implemented efficiently.

We have extended GLUE to build CGLUE, a system that employs the above beam search solution to find complex mappings. While CGLUE exploits information in the data and the taxonomic structures for matching purposes, it has not yet exploited domain constraints (and so does not use relaxation labeling). In Section 8.4 we briefly discuss future work on exploiting domain constraints. In what follows we describe experiments with the current CGLUE system.

8.2 Empirical Evaluation

We have evaluated CGLUE on three real-world domains, whose characteristics are shown in Table 3. The first domain is "Course Catalog I" that we used in our GLUE experiments for 1-1 matching. This domain was described in Table 2 and reproduced in Rows 1-2 of Table 3. We found that this domain has a fair number of complex mappings (7-11 out of 34-39

Table 3 Domains and taxonomies for experiments with CGLUE.

Taxonomies		# nodes	# non-leaf nodes	depth	# instances in taxonomy	max # instances at a leaf	max # children of a node	# manual mappings created		
								complex	1-1	total
Course Catalog I	*Cornell*	34	6	4	1526	155	10	11	23	34
	Washington	39	8	4	1912	214	11	7	32	39
Company Profiles I	*Standard*	48	10	3	2441	353	10	7	41	48
	Yahoo	22	6	3	2461	656	12	9	13	22
Company Profiles II	*Standard*	248	23	3	11079	557	24	20	228	248
	Yahoo	95	11	3	8817	656	25	43	3	46

mappings), and that we could find the correct complex mappings fairly quickly. The domain therefore is well-suited for our purpose.

In contrast, we found that domain "Company Profiles" for the 1-1 matching case (Table 2) contains few complex mappings and that the correct complex mappings were extremely difficult to detect. Without knowing the correct complex mappings (i.e., the "gold standard"), however, we would not be able to evaluate CGLUE.

Therefore, we modified the domain so that we can find the set of all correct complex mappings. Our goal is to use these mappings to evaluate the mappings that CGLUE returns. We removed and merged certain nodes, and created two smaller versions – "Company Profiles I" and "Company Profiles II", which are described in Rows 3-6 of Table 3. The latter domain is much larger than the former (95-248 nodes vs. 22-48). Both of them contain a fair number of complex mappings (7-43).

Similar to the 1-1 matching case, we chose to evaluate CGLUE using the *Jaccard* similarity measure. Given this measure, we manually created the correct mappings between the taxonomies. The last three columns of Table 3 show the number of complex and 1-1 mappings (and the total number of mappings) that we created for each taxonomy. The domains and manual mappings will be made available at the *Illinois Semantic Integration Archive* (http://anhai.cs.uiuc.edu/archive).

8.3 Matching Accuracy

For each domain, we applied CGLUE to find semantic mappings. For "Course Catalog I", for example, we applied CGLUE to find mappings from *Washington* to *Cornell*, then from *Cornell* to *Washington*. Thus for the three domains we have a total of six matching scenarios.

Accuracy for Complex Mappings: Figure 8.a shows the matching accuracies for the six scenarios. These accuracies *were evaluated on complex mappings only, excluding 1-1 mappings*. Consider the first scenario, *W2C* (shorthand for "from *Washington* to *Cornell*"), which has four accuracy bars. The first bar shows the percentage of complex mappings that CGLUE predicted correctly. Specifically, it says that CGLUE correctly produced 57% of complex mappings for *Washington* (4 out of 7). We will explain the meaning of the remaining three bars shortly.

For now, focusing on the first accuracy bars of the six matching scenarios, we can draw several conclusions. First, CGLUE achieved accuracy 50-57% on half of the matching scenarios: the *W2C* and the two *S2Y* ones. This is significant considering that each complex mapping involves 4-5 nodes and yet CGLUE managed to predict these nodes correctly in more than half of the cases, choosing from a very large pool of mapping candidates.

Second, CGLUE did not do as well on the remaining three scenarios, achieving accuracy of 16-27%. Upon close examination, we found that in each of these scenarios, there were several "errant" nodes that appeared in numerous predictions made by CGLUE, thus rendering these predictions incorrect. For example, in the *C2W* scenario, the node Greek-Courses appears in 45% of the complex mappings made by CGLUE. Such nodes appear to contain very little or vacuous data, leaving little room for learning techniques to classify them correctly. We observed that "errant" nodes can be easily detected by the user from a quick inspection of the mappings produced by CGLUE. Once detected, they can be removed and CGLUE can be rerun to produce more accurate mappings. Indeed, for the above three matching scenarios, after detecting "errant" nodes (we currently define these nodes to be those that appear in more than 40% of the mappings), removing them, and reapplying CGLUE, we obtained accuracies of 50-51%, an improvement of 23-29% over the initial accuracies.

Relaxing the Notion of Correct Matching: While experimenting, we observed that our definition of matching accuracy is in fact a *pessimistic* estimation of the usefulness of CGLUE. Suppose the correct mapping for node A is $A = (B \cup C \cup D)$. Then CGLUE may predict $A = (B \cup C \cup E)$, which we so far have discarded as incorrect. However, often when CGLUE produces such a mapping, the user can immediately tell (from the names of the nodes) that B and C should be included in a mapping for A, and that E should be excluded. Thus, even a partially correct mapping such as the one above could prove very useful for the user.

To examine the extent to which CGLUE produces partially correct mappings, we consider looser notions of correctness. Suppose that the correct (manual) mapping for A is the set of nodes M_c and that CGLUE predicts the set of

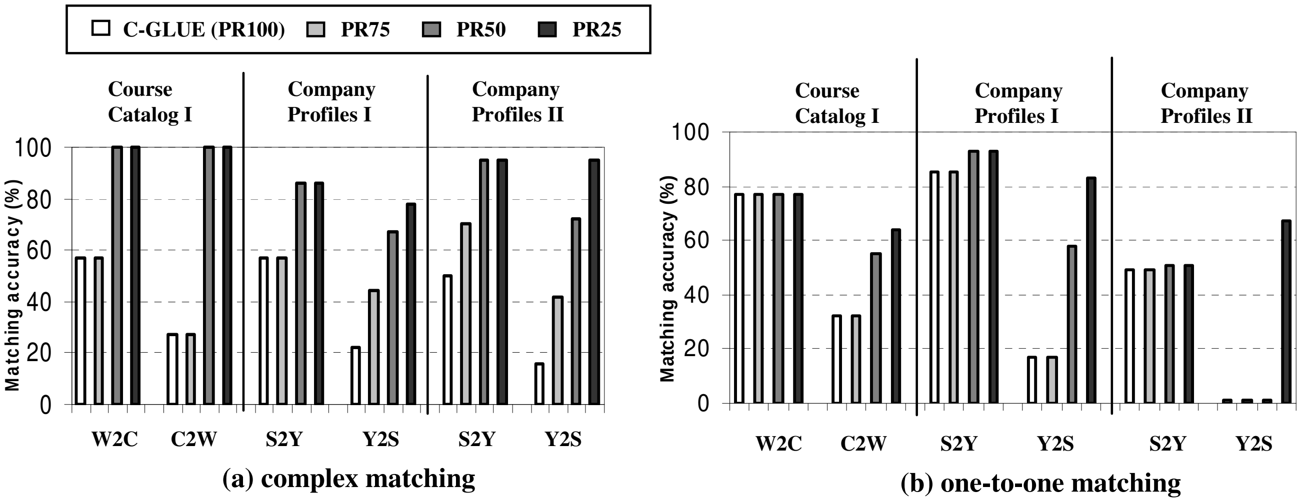

Fig. 8 Matching accuracy of CGLUE.

nodes M_p. We define the *precision* of this prediction to be $|M_p \cap M_c|/|M_p|$, and its *recall* to be $|M_p \cap M_c|/|M_c|$. Then we say that under correctness level t, a predicted mapping is correct if both of its precision and recall are greater or equal to $t\%$. We use "PRt" to refer to the matching accuracy that is computed using correctness level t.

Returning to Figure 8.a, we have discussed the first bar of each matching scenario, which corresponds to accuracy level PR100. The remaining three bars of each scenario correspond to accuracy levels PR75, PR50, and PR25, respectively. As can be seen, excluding the 50-57% of mappings that CGLUE predicted correctly (as we discussed earlier), CGLUE also was partially correct for an overwhelming majority of remaining mappings. At PR25, CGLUE was partially correct for 90-100% of the remaining mappings.

Accuracy for 1-1 Mappings: Since CGLUE can mistakenly issue complex-mapping predictions for nodes whose correct mappings are 1-1, we wanted to know how well CGLUE makes predictions for such nodes. Figure 8.b shows matching accuracies in a way similar to that of Figure 8.a, except that here the accuracies are evaluated over the 1-1 mappings. For example, the first bar of this figure says that out of 32 1-1 mappings of taxonomy *Washington* (see Table 3), CGLUE correctly predicted 25, achieving an accuracy of 78%.

As can be seen from the figure, CGLUE achieves high accuracy in half of the matching scenarios (*W2C* and the two *S2Y*s), ranging from 50-85%. It achieves lower accuracies of 0-35% in the remaining scenarios. (Though the accuracy 0% of the last *S2Y* scenario should be discounted because here we have only three 1-1 mappings; excluding this scenario the accuracy is 17-35%.) Again, this low accuracy is largely due to the fact that several "errant" nodes appear in numerous mappings, rendering them incorrect. Removing these "errant" nodes yields accuracies 46-52%, thus resulting in an improvement of 17-29%.

Figure 8.b further shows that at PR25 CGLUE achieves accuracy of 52-84%. By definition, any prediction that CGLUE makes that is correct at PR25 would contain at most four nodes and *must contain the correct matching node*. As such, the prediction would be useful to the user, because he or she often could quickly identify the correct matching node. Thus, the above result is significant because it suggests that CGLUE could help the user locate the correct node for 52-84% of the 1-1 mappings.

8.4 Discussion

The above experiments show that with the current simple solution that uses beam search, CGLUE already achieves good results for both 1-1 and complex matching. These results can be improved in a variety of ways, one of which is to incorporate domain constraints. For example, we observed that many mappings made by CGLUE include semantically unrelated nodes, such as "Oil-Utilities = Oil-Equipments-Companies ∪ Food-Companies". Clearly, if we can exploit the constraint "concept Oil-Utilities is semantically unrelated to Food-Companies", we should be able to "clean" the above mapping by removing the node Food-Companies, thus improving the overall matching accuracy.

We now discuss removing the assumption that the children of any taxonomic node are mutually exclusive and exhaustive. Without this assumption we must consider the space of candidates that are built using both union and difference operators. Our beam-search approach can be extended to handle the difference operator. The only key difficulty is in the implementation of Step 2.a of the algorithm in Figure 7.

Consider a mapping candidate that is the difference of two nodes B and C. Step 2.a computes the similarity between this candidate and the input node A. This can be done only if we can compute the difference between B and C, which in turn requires solving the *object identification* problem: deciding if any two given instances from B and C match. Object identification is a long-standing and difficult problem in databases and AI. We note that this problem is not peculiar to our approach. Indeed, it appears that any satisfactory solution to complex matching for taxonomies must address this problem.

In many specialized cases, the object identification problem can be solved by exploiting domain regularities. For example, in "company profiles" domains we can infer that two companies match if their urls match. In the "course catalog" domains two courses match if the sets of their course ids overlap. In such cases, our beam-search solution can be implemented without any difficulty.

Finally, we note that CGLUE (and in fact the vast majority of automatic ontology/schema matching tools) only *suggests* mappings to the user. Developing techniques to help the user efficiently post-process such suggested mappings to arrive at the final correct mappings would be an interesting and important topic for future research.

9 Related Work

We now describe related work to GLUE from several perspectives.

Ontology Matching: Many works have addressed ontology matching in the context of ontology design and integration (e.g., [Cha00, MFRW00, NM00, MWJ99]). These works do not deal with explicit notions of similarity. They use a variety of heuristics to match ontology elements. They do not use machine learning and do not exploit information in the data instances. However, many of them [MFRW00, NM00] have powerful features that allow for efficient user interaction, or expressive rule languages [Cha00] for specifying mappings. Such features are important components of a comprehensive solution to ontology matching, and hence should be added to GLUE in the future.

Several recent works have attempted to further automate the ontology matching process. The Anchor-PROMPT system [NM01] exploits the general heuristic that paths (in the taxonomies or ontology graphs) between matching elements tend to contain other matching elements. The HICAL system [RHS01] exploits the data instances in the overlap between the two taxonomies to infer mappings. [LG01] computes the similarity between two taxonomic nodes based on their signature TF/IDF vectors, which are computed from the data instances.

Schema Matching: Schemas can be viewed as ontologies with restricted relationship types. The problem of schema matching has been studied in the context of data integration and data translation (e.g., [DR02, BM02, EJX01, CHR97, RS01], see also [RB01] for a survey). Several works [MZ98, MBR01, MMGR02] have exploited variations of the general heuristic "two nodes match if nodes in their neighborhood also match", but in an isolated fashion, and not in the same general framework we have in GLUE.

GLUE is related to LSD, our previous work on schema matching [DDH01]. LSD illustrated the effectiveness of multi-strategy learning for schema matching. However, it assumes that we can use a set of *manually* given mappings on several sources as training examples for learners that predict mappings for subsequent sources. In GLUE since our problem is to match a pair of ontologies, there are no manual mappings for training, and we need to obtain the training examples for the learner automatically. Further, since GLUE deals with a more expressive formalism (ontologies versus schemas), the role of constraints is much more important, and we innovate by using relaxation labeling for this purpose. Finally, LSD did not consider in depth the semantics of a mapping, as we do here.

Notions of Similarity: The similarity measure in [RHS01] is based on κ statistics, and can be thought of as being defined over the joint probability distribution of the concepts involved. In [Lin98] the authors propose an information-theoretic notion of similarity that is based on the joint distribution. These works argue for a single best universal similarity measure, whereas GLUE allows for application-dependent similarity measures.

Ontology Learning: Machine learning has been applied to other ontology-related tasks, most notably learning to construct ontologies from data and other ontologies, and extracting ontology instances from data [Ome01, MS01, PRV01]. Our work here provides techniques to help in the ontology construction process [MS01]. [Mae01] gives a comprehensive summary of the role of machine learning in the Semantic Web effort.

1-1 and Complex Matching: The vast majority of current works focus on finding 1-1 semantic mappings. Several works (e.g., [MZ98]) deal with complex matching in the sense that such matchings are hard-coded into rules. The rules are systematically tried on the elements of given representations, and when such a rule fires, the system returns the complex mapping encoded in the rule. The Clio system [MHH00, YMHF01, PVH+02] creates complex mappings for relational and XML data. Clio however relies heavily on user interaction and does not use machine learning techniques. Thus, our work with CGLUE is in a sense complementary to that of Clio.

10 Conclusion and Future Work

With the proliferation of data sharing applications that involve multiple ontologies, the development of automated techniques for ontology matching will be crucial to their success. We have described an approach that applies machine learning techniques to match ontologies. Our approach, as embodied by the GLUE system, is based on *well-founded notions of semantic similarity*, expressed in terms of the joint probability distribution of the concepts involved. We described the use of machine learning, and in particular, of *multi-strategy learning*, for computing concept similarities.

We introduced relaxation labeling to the ontology-matching context, and showed that it can be adapted to efficiently *exploit a variety of heuristic knowledge and domain-specific* *constraints* to further improve matching accuracy. Our experiments showed that GLUE can accurately match 66 - 97% of the nodes on several real-world domains. Finally, we have extended GLUE to build CGLUE, a system that *finds complex mappings between ontologies*. We described experiments with CGLUE that show the promise of the approach.

Aside from striving to improve the accuracy of our methods, our main line of future research involves extending our techniques to handle more sophisticated mappings between ontologies, such as those involving attributes and relations.

Acknowledgments: We thank Phil Bernstein, Geoff Hulten, Natasha Noy, Rachel Pottinger, Matt Richardson, Pradeep Shenoy, and the reviewers for their invaluable comments. This work was supported by NSF Grants 9523649, 9983932, IIS-9978567, IIS-9985114, a UIUC Start-Up Grant, and an NCSA Research Assistantship. Pedro Domingos is also supported by an IBM Faculty Patnership Award. Alon Halevy is also supported by a Sloan Fellowship and gifts from Microsoft Research, NEC and NTT. Part of this work was done while AnHai Doan was at the University of Washington.

References

[Agr90] A. Agresti. *Categorical Data Analysis*. Wiley, New York, NY, 1990.

[BG00] D. Brickley and R. Guha. Resource Description Framework Schema Specification 1.0, 2000.

[BKD+01] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling knowledge representation on the Web by Extending RDF Schema. In *In Proceedings of the Tenth Int. World Wide Web Conference*, 2001.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 279, 2001.

[BM02] J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *Proceedings of the Conf. on Advanced Information Systems Engineering (CAiSE)*, 2002.

[CDI98] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced Hypertext Categorization Using Hyperlinks. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[CGL01] D. Calvanese, D. G. Giuseppe, and M. Lenzerini. Ontology of Integration and Integration of Ontologies. In *Proceedings of the 2001 Description Logic Workshop (DL 2001)*, 2001.

[Cha00] H. Chalupsky. Ontomorph: A translation system for symbolic knowledge. In *Principles of Knowledge Representation and Reasoning*, 2000.

[CHR97] C. Clifton, E. Housman, and A. Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *Proc. of the IFIP Working Conference on Data Semantics (DS-7)*, 1997.

[dam] www.daml.org.

[DDH01] A. Doan, P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. In *Proceedings of the ACM SIGMOD Conference*, 2001.

[DMDH02] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map ontologies on the Semantic Web. In *Proceedings of the World-Wide Web Conference (WWW-02)*, 2002.

[DMDH03] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Ontology matching: A machine learning approach. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*. Springer-Velag, 2003.

[Doa02] A. Doan. Learning to map between structured representations of data, 2002. PhD thesis, University of Washington, http://anhai.cs.uiuc.edu/home/thesis.html.

[DP97] P. Domingos and M. Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29:103–130, 1997.

[DR02] H. Do and E. Rahm. Coma: A system for flexible combination of schema matching approaches. In *Proceedings of the 28th Conf. on Very Large Databases (VLDB)*, 2002.

[EJX01] D. Embley, D. Jackman, and L. Xu. Multifaceted exploitation of metadata for attribute match discovery in information integration. In *Proceedings of the WIIW Workshop*, 2001.

[Fen01] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2001.

[goo] www.google.com.

[HH01] J. Heflin and J. Hendler. A Portrait of the Semantic Web in Action. *IEEE Intelligent Systems*, 16(2), 2001.

[HZ83] R.A. Hummel and S.W. Zucker. On the Foundations of Relaxation Labeling Processes. *PAMI*, 5(3):267–287, May 1983.

[iee01] *IEEE Intelligent Systems*, 16(2), 2001.

[LG01] M. Lacher and G. Groh. Facilitating the exchange of explicit knowledge through ontology mappings. In *Proceedings of the 14th Int. FLAIRS conference*, 2001.

[Lin98] D. Lin. An Information-Theoritic Definiton of Similarity. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1998.

[Llo83] S. Lloyd. An optimization approach to relaxation labeling algorithms. *Image and Vision Computing*, 1(2), 1983.

[Mae01] A. Maedche. A Machine Learning Perspective for the Semantic Web. Semantic Web Working Symposium (SWWS) Position Paper, 2001.

[MBR01] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[MFRW00] D. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera Ontology Environment. In *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000.

[MHH00] R. Miller, L. Haas, and M. Hernandez. Schema mapping as query discovery. In *Proc. of VLDB*, 2000.

[MMGR02] S. Melnik, H. Molina-Garcia, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2002.

[MS01] A. Maedche and S. Staab. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2), 2001.

[MWJ99] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic Integration of Knowledge Sources. In *Proceedings of Fusion'99*, 1999.

[MZ98] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1998.

[NM00] N.F. Noy and M.A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.

[NM01] N.F. Noy and M.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)*, 2001.

[Ome01] B. Omelayenko. Learning of Ontologies for the Web: the Analysis of Existent approaches. In *Proceedings of the International Workshop on Web Dynamics*, 2001.

[ont] http://ontobroker.semanticweb.org.

[owl] http://www.w3.org/tr/owl-ref.

[Pad98] L. Padro. A Hybrid Environment for Syntax-Semantic Tagging, 1998. PhD thesis, Universitat Polit'ecnica de Catalunya (UPC).

[PRV01] N. Pernelle, M-C. Rousset, and V. Ventos. Automatic Construction and Refinement of a Class Hierarchy over Semi-Structured Data. In *The IJCAI Workshop on Ontology Learning*, 2001.

[PVH$^+$02] L. Popa, Y. Velegrakis, M. Hernandez, R. J. Miller, and R. Fagin. Translating web data. In *Proc. of the 28th Int. Conf. on Very Large Databases (VLDB-02)*, 2002.

[RB01] E. Rahm and P.A. Bernstein. On matching schemas automatically. *VLDB Journal*, 10(4), 2001.

[RHS01] I. Ryutaro, T. Hideaki, and H. Shinichi. Rule Induction for Concept Hierarchy Alignment. In *Proceedings of the 2nd Workshop on Ontology Learning at the* 17^{th} *Int. Joint Conf. on AI (IJCAI)*, 2001.

[RS01] A. Rosenthal and L. Seligman. Scalability issues in data integration. In *Proceedings of the AFCEA Federal Database Conference*, 2001.

[TW99] K. M. Ting and I. H. Witten. Issues in stacked generalization. 10:271–289, 1999.

[Usc01] M. Uschold. Where is the semantics in the Semantic Web? *Submitted for publication*, 2001.

[vR79] van Rijsbergen. *Information Retrieval*. London:Butterworths, 1979. Second Edition.

[Wol92] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[YMHF01] L.L. Yan, R.J. Miller, L.M. Haas, and R. Fagin. Data Driven Understanding and Refinement of Schema Mappings. In *Proceedings of the ACM SIGMOD*, 2001.