Enabling Domain-Awareness for a Generic Natural Language Interface

Yunyao Li[‡*]   **Ishan Chaudhuri**[‡]   **Huahai Yang**[§]   **Satinder Singh**[‡†]   **H. V. Jagadish**[‡*]

[‡]University of Michigan
Ann Arbor, MI 48109
{yunyaol,ishanrc,baveja,jag}@umich.edu

[§]University at Albany, SUNY
Albany, NY 12222
hyang@albany.edu

Abstract

In this paper, we present a learning-based approach for enabling domain-awareness for a generic natural language interface. Our approach automatically acquires domain knowledge from user interactions and incorporates the knowledge learned to improve the generic system. We have embedded our approach in a generic natural language interface and evaluated the extended system against two benchmark datasets. We found that the performance of the original generic system can be substantially improved through automatic domain knowledge extraction and incorporation. We also show that the generic system with domain-awareness enabled by our approach can achieve performance similar to that of previous learning-based domain-specific systems.

Introduction

This work is intended to achieve the promise shown by NaLIX (Natural Language Interface to XML), a generic natural language interface for an XML database (Li 2005; 2006a). NaLIX can accept an arbitrary English language sentence as a query input. This query, which can include complex query semantics such as aggregation, nesting, and value joins, is then translated into an XQuery expression. The core system has limited linguistic capabilities. Whenever the system cannot properly translate a given query, it sends meaningful feedback to the user and asks her to reformulate the query into one that the system can understand. A previous study (Li 2006a) has shown that such a system is already usable in practice—users can issue complex database queries in plain English and obtain precise results back.

While generic systems like NaLIX are widely praised for their portability—they can be universally deployed without expensive efforts for building domain knowledge bases—their portability also comes at a price. For instance, in NaLIX, reformulations are often required for queries containing domain-specific terms; additionally, terms with important domain semantics may simply be ignored by the generic system, resulting in loss of accuracy.

In this paper, we describe our approach to enable domain-awareness for such a generic natural language interface to improve its translation accuracy and reduce the need for reformulation without losing its portability. Whereas much

research has been done on the use of learning to build a domain-specific natural language interface for databases (NLIDB), we believe ours is the first attempt to extend a generic NLIDB by automatically extracting domain knowledge from user interaction without explicitly burdening the users. Below, we briefly summarize related research, and then describe a motivational example, followed by our approach for extracting domain knowledge from user interactions and incorporating domain knowledge into a generic system. Finally, we report on experiments that evaluate our approach against two benchmark datasets.

Related Work

Mapping a given sentence into a structured data query or a general logical form is a central problem in designing natural language interface to databases (NLIDB). Extensive research has been done on NLIDBs. A comprehensive summary of earlier efforts is provided by (Androutsopoulos 1995). Recent research in this area has focused on leveraging advances in parsing techniques to design generic systems with easy portability (Popescu 2003; Li 2006a). Our approach can easily be adapted by such systems to enable domain-awareness.

Previous learning-based NLIDBs (Androutsopoulos 1995), including recent efforts by Tang & Mooney (2001) and Zettlemoyer & Collins (2005), study how to learn the mapping from sentences to logical forms. Expensive training is required to learn such mapping rules, making it difficult to adapt these systems to a new domain. In contrast, the system with domain-awareness enabled by our approach retains the portability of the original generic system and thus requires no domain knowledge to be used in a new domain. Furthermore, previous systems require expensive manual creation of training data and demand expertise in both a specific domain and a formal query language. By comparison, building a knowledge base using our approach is much easier—no expertise in any formal query language is required: query pairs can easily be obtained as training data from actual user interaction with the system. Finally, unlike previous systems, where new training data typically cannot be exploited without re-training of the entire system, our approach allows incremental accumulation of the knowledge base.

With the rise of semantic Web, growing attention has been paid to domain-aware systems, with the focus on the learning of ontology (Maedche & Staab 2001; Xu 2002; Gómez-pérez & Manzano-macho 2004; Buitelaar 2005). We view these learning approaches and our method as complementary to each other: while a generic system can take


Table 1: Sample Types of Tokens and Marker

Type	Semantic Contribution	Description
Command Token(CMT)	Return Clause	Top main verb or wh-phrase (Quirk 1985) of parse tree, from an enum set of words and phrases
Operator Token(OT)	Operator	A phrase from an enum set of preposition phrases
Value Token(VT)	Value	A noun or noun phrase in quotation marks, a proper noun or noun phrase, or a number
Name token(NT)	Basic Variable	A non-VT noun or noun phrase
Connection Marker(CM)	Connect two related tokens	A preposition from an enumerated set, or non-token main verb
Modifier Marker(MM)	Distinguish two NTs	An adjectives as determiner or a numeral as predetermine or postdeterminer
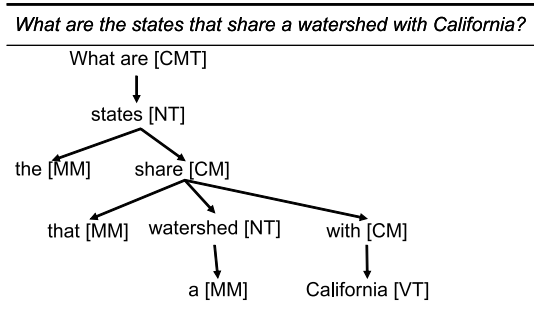
(a) Invalid Query with a Domain-Specific Term
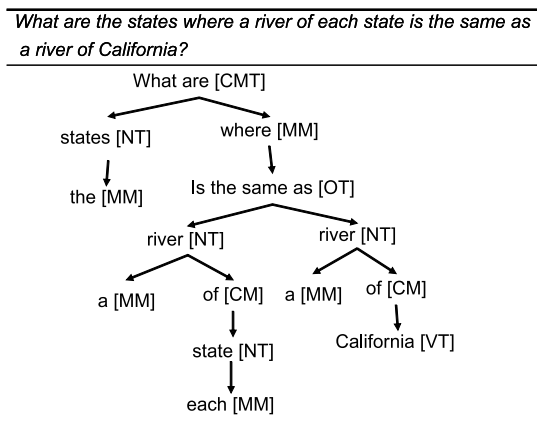
(b) Valid Query after Reformulation

Figure 1: Sample Iteration For Domain Knowledge Learning.

advantage of such learning methods to improve domain-awareness through the use of ontology-based term expansion, our learning approach can provide valuable training data for these methods.

Approach

Our approach to enable domain-awareness is implemented by extending the original NaLIX system with automatic domain knowledge extraction and exploitation based on a generic domain knowledge representation.

Background and Motivation

NaLIX consists of three main components for query translation: (a) given the dependency parse tree of a given English sentence, the *classifier* classifies words/phrases that can be mapped into XQuery components as tokens and those that

cannot as markers (sample types are listed in Table 1); (b) the *validator* then determines whether the parse tree is one that the system knows how to map into XQuery; **if it is**, (c) the *translator* translates the parse tree into an XQuery expression; **else**, (d) the *message generator* sends feedback to the user and asks for reformulation. For a more detailed description of NaLIX, the reader is referred to (Li 2006a).

NaLIX is purely generic and thus may not be able to correctly interpret terms with domain-specific semantics. Consider a geographic database with information about states and the rivers that run through their borders. Given the query "*What are the states that share a watershed with California?*", the system cannot properly interpret "*watershed*," as it is neither a generic term nor a term that can be found in the database. The system then sends feedback to the user and requests him to rephrase the query.[1] The user might then respond by submitting "*What are the states where a river of each state is the same as a river of California?*", which contains no domain specific term.

Such an iteration results in a pair of queries with equivalent semantics, where the first requires domain knowledge and the second does not, providing a valuable opportunity for learning new domain knowledge. Such pairs of queries can be obtained automatically as a side effect of query reformulation by users. The goal of our approach is to take advantage of such query pairs and allow the system to learn from user actions while they are doing their normal work and thus become more responsive to users over time.

To enable domain-awareness for NaLIX, we added the *knowledge extractor* and *knowledge base* to extract domain knowledge from user interactions and store the knowledge respectively. We also added the *domain adapter* to determine and apply applicable domain knowledge on a classified parse tree before sending it to the *validator*. The resulting system is Domain-aware NaLIX(DaNaLIX).

Knowledge Representation

The knowledge base must capture useful domain-specific semantics that can be used to improve query translation in a generic NLIDB. The model for knowledge representation needs to be generic enough to be able to capture such domain knowledge for any given domain. It should also be able to exploit what the generic NLIDB can already do—mapping domain-independent knowledge to query semantics. To do so, we choose a simple term mapping form, which expresses domain-specific knowledge in generic terms, over complex semantic logical forms such as lambda-calculus (Barendregt 1984). Specifically, we represent domain knowledge as a set of rules that can be used to transform the parse tree of a sentence that contains terms with domain-specific semantics into one that does not. The *validator* and *translator*

---

[1]Details of such feedback in NaLIX can be found in (Li 2006a).

Algorithm 1: `ApplyRule(Node n, Rule r)`

```
// check whether the left-hand side of
   the rule matches the
// tree rooted at n
if treeMatches(r.leftSide(),n) then
    // all matching conditions are
       satisfied
    // transform the subtree rooted at n
    return transformTree(n,r.rightSide());
else
    foreach child in n.children() do
        // search the rest of the tree for
           matches
        applyRule(child,r);
```


(a) Visualization

Source Tree		Target Tree
return (state (share, (watershed, with ($[id_0]$))))	⇒	**return** (states, where (**equal** (river (of (state)) river (of ($[id_0]$))))

(b) Shorthand Representation

Figure 2: Sample Transformation Rule.
Symbol ⇒ stands for "transform into." **return** and **equal** in (b) correspond to [CMT] and (*is*, *same as*) in (a) respectively; $[id_0]$ in (b) corresponds to [VT] in (a); markers such as *the*, *a* in (a) are not included in (b) for the purpose of presentation.

Figure 3: Data Structure of Source Tree Nodes in a Transformation Rule. The hollow triangle shape indicates *inheritance* relationship between the two types of nodes.

can then operate on the transformed tree using only domain-independent knowledge.

Figure 2 shows an example of transformation rules. Each transformation rule is composed of two parts: a *source tree* and a *target tree*. The source tree and target tree in each rule are semantically equivalent, but the source tree contains terms with domain-specific meanings (e.g., "*watershed*"), while the target tree does not.
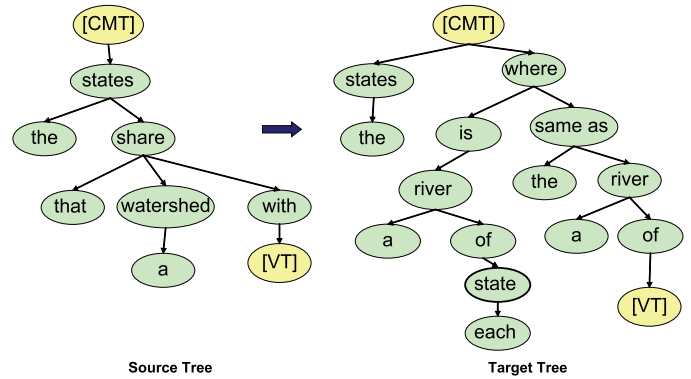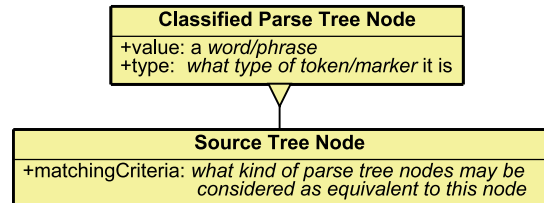
Figure 3 depicts the data structure of source tree nodes in a transformation rule.[2] In addition to the value and type of its corresponding classified parse tree node, a source tree node also contains information indicating how this node should be matched during transformation (denoted as `matchCriteria`). Each node is assigned a default `matchCriteria` value based on its node type and position in the tree. For example, the default `matchCriteria` value for root node (e.g., the node labeled as [CMT] in Figure 2(a)) in the transformation rule is `byType`.

In this paper, we only consider domain knowledge in the form of pairwise equivalence that can improve the domain-awareness for the generic NLIDB. Other forms of domain knowledge (e.g., *is-a-part-of* relationship), although useful in generating suitable query relaxation, are not considered, because issues related to such domain knowledge are universal regardless of whether a query is written in English or in a formal database language. In addition, one may argue that a better representation for pairwise equivalence (e.g., $A \Rightarrow B$ and $A \Rightarrow C$) is in the form of a set of equivalences (e.g., $eq(A, B, C)$). While such an argument may be true in theory, the simple pairwise representation is more suitable for enabling domain-awareness of a generic NLIDB. First, knowing $A \Rightarrow B$ and $A \Rightarrow C$ is already adequate to allow the system to correctly interpret A; knowing $B \Rightarrow C$ will not contribute to the translation. More importantly, since in our approach rules are incrementally learned from user interactions, it is unlikely that $A \Rightarrow B$ and $A \Rightarrow C$ will coexist—once $A \Rightarrow B$ is created, it will be applied for any later queries and thus $A \Rightarrow C$ will not be created unless B and C are not deemed as equivalent by different users.

Domain Knowledge Learning
Algorithm 2 shows how we extract domain knowledge from user interaction with the system. Consider a pair of queries in the form of classified parse trees (denoted as S and T respectively), where S is the parse tree of a query that the

---

[2]The data structure for target tree nodes is the same as that for classified parse tree nodes.

system was unable to process without domain knowledge, and T is the parse tree for the query that requires no domain knowledge to be correctly translated. Our algorithm begins with the roots of the parse trees: s (root of S) and t (root of T). We then recursively traverse these two trees in parallel starting from the roots. Two nodes for each tree are considered *equivalent* if their parents are equivalent, and each of their corresponding children have the same type and value. If two nodes from S and T are found to be not equivalent, we create a new rule, denoted $rule$, and add these two nodes to it. We then add nodes[3] in the subtrees of these nodes to $rule$. The creation of $rule$ does not stop until we have found two nodes with identical types, values and subtrees[4] or we have already traversed the entire subtrees. The algorithm continues to traverse the tree until all pairs of non-equivalent nodes have been found. Multiple rules may be found for a given

---

[3]Whenever a node added to the source tree of the rule, it is automatically assigned a default `matchCriteria` value based on its type and position in the source tree. Details are omitted for simplicity.
[4]Nodes with subtrees composed solely of markers, such as "the" or "a," are not considered for stop condition.

Algorithm 2: `ExtractKnowledge(Node s,Node t)`

```
// addSubtreesToRule(s,t) recursively adds
   nodes from
// the subtrees rooted at s and t until
   nodes with identical
// subtrees are found

// Learn a rule if the children do not
   match
if isNonEquivalent(s,t) then
    rule ← addSubtreesToRule(s,t);
    return rule ;
else
    // traverse the trees in parallel
    childPairs ← (s.children(),t.children());
    foreach (s,t) in childPairs do
        // find non-equivalent nodes under
           s and t
        rules ← rules + extractKnowledge(s,t);
    return rules
```

What are the states that share a watershed with New York, ordered by their population?

What are [CMT]

states [NT]   where [MM]

the [MM]   Is the same as [OT]

river [NT]   river [NT]

a [MM]   of [CM]   a [MM]   of [CM]

state [NT]   New York [VT]

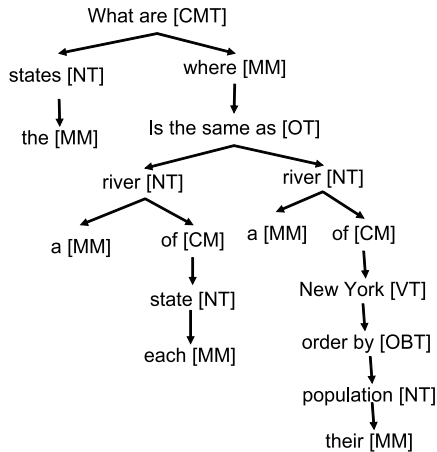each [MM]   order by [OBT]

population [NT]

their [MM]

Figure 4: Sample Knowledge Incorporation.
The parse tree after transformation based on the rule illustrated in Figure 2.

pair of queries. Each rule maps a source tree (a partial tree from S) to a target tree (a partial tree form T).

Figure 1 illustrates an iteration through which DaNaLIX learns the sample rule shown in Figure 2. We can find that the roots of the classified parse trees in Figure 1(a) and Figure 1(b) are not equivalent, as they have different children. Based on Algorithm 2, nodes in their subtree are added to a rule until the stop condition is met. In this particular case, such nodes are the last nodes of the trees, the value tokens "*California.*" The two nodes are added to the target tree and the source tree respectively, with their `matchCriteria` values set to be `byValue`, thus denoted as [VT] by their types in Figure 2(a). Since we have already traversed all the nodes at this point, the algorithm stops and returns the rule. The result is a rule that captures the meaning of the domain-specific term "*watershed*" in the context of *state*s and *river*s as shown in Figure 2.

Domain Knowledge Incorporation
Algorithm 1 presents how the *domain adapter* uses a rule to transform a given query. It begins by traversing the parse tree of the query until it finds a portion of the tree that matches the source tree specified by the left-hand side of the rule (based on the `matchCriterias` of the source tree nodes). The *domain adapter* then replaces this portion of parse tree with the target tree specified by the right-hand side of the rule.

In our example of a geographic database, consider the following query submitted after the iteration illustrated in Figure 1—"*What are the states that share a watershed with New York, ordered by their population?*" The *domain adapter* can find that the rule (Figure 2) learned from the iteration matches the query. One may notice that the value of [VT] in original queries in Figure 1 is "*California,*" which is not contained by the new query. Since the `matchCriteria` for [VT] is `byType`, [VT] is still considered as matching with "*New York,*" which is also a value token. The parse tree after the transformation is shown in Figure 4.

Given a query with terms that have domain-specific meanings, DaNaLIX searches the *knowledge base* for matching rules. If one is found, DaNaLIX uses it to transform the original parse tree and returns results based on the parse tree after transformation. If more than one rule is found, a randomly selected rule from applicable rules with the highest confidence score[5] is used to transform the query. The system then informs the user about this transformation and gives him the option of rejecting the rule used, or processing the query with another suitable rule. The confidence score of a rule will be decreased for rejections and increased for selections. If the user does not reject the rule or attempt to rephrase the query, the rule is considered as selected. Rules with sufficiently low confidence score will be eliminated.

Experimental Evaluation
To evaluate the effectiveness of our approach, we conducted an experiment comparing NaLIX and its domain-aware enhancement, DaNaLIX. In addition, we compare these systems with two notable learning-based NLIDBs, COCKTAIL (Tang & Mooney 2001) and GENLEX (Zettle-moyer & Collins 2005), both mapping natural language questions to semantic representations. For our experiment, we chose the datasets on which COCKTAIL and GENLEX have previously reported results.

Methods
Data Set The experiment uses datasets from two domains (Tang & Mooney 2001): **Geo880**, a set of 880 queries to a database of U.S. geography, and **Jobs640**, a set of 640 queries to a database of job listings. Both databases are originally tagged in Prolog-style; we converted them into XML format to be used by NaLIX and DaNaLIX.

The main goal of our experiment is to evaluate the improvement of DaNaLIX over NaLIX. As mentioned earlier, NaLIX only accepts queries satisfying a specific grammar derived from XQuery semantics. For instance, NaLIX requires a query to explicitly specify the object(s) to be returned. A query that fails to do so (e.g., a *yes/no* question) will need to be rewritten. Such a reformulation demands only grammatical changes and has been found to be relatively trivial for users to do, especially with the help of system suggestions (Li 2006b). To be able to make a fair comparison between DaNaLIX and NaLIX, we first examined

---
[5]Every rule is initially assigned with the same default confidence score.

Table 2: Sample Ambiguous Queries

- I wonder what JPL does on Unix with Prolog and Vax ?
- Prolog AI and LISP and Graphics?
- Vanity wants 5000 a month with buzzwords like Java Apple Internet and California?

queries in the data set to ensure that queries in the experiment were already NaLIX-acceptable; for those that are not, we manually made the necessary grammatical changes.

Our learning algorithm requires pairs of semantically equivalent natural language queries as input for rule extraction. Both **Geo880** and **Jobs640** provide only English queries and their corresponding Prolog-style semantic representations. To create training data for DaNaLIX, semantic representation of queries in the datasets were converted into equivalent natural language queries that can be correctly translated by NaLIX. To reduce the bias resulted from one person's natural language formulation, the datasets were randomly divided in half, and each half was assigned to a student involved in this project for independent conversion. The converted queries were randomly mixed up, and paired with their corresponding original queries. Each such pair of queries was then used as either a training sample or test data in the experiment.

Eight queries in the **Jobs640** dataset are highly ambiguous, as can be seen from the samples listed in Table 2: these queries are excluded from our experiment. Additionally, NaLIX does not support mathematical calculations. As such, we changed any query requiring mathematical calculation into one that returns the needed information for the calculation. For example, the query "*What is the density of Texas?*" is changed into "*What is the population and area of Texas?*", where $density = population/area$. Queries asking for aggregated information over the results of calculation, e.g., "*What is the state with the smallest density?*", cannot be supported without the actual calculation. Such queries (totally 35 out of 880 from **Geo880**) were excluded from our experiment.

Measurement The evaluation metrics we used were *precision* and *recall*, defined as $precision = \frac{\# \, correct \, queries}{total \, \# \, parsed \, queries}$ and $recall = \frac{\# \, correct \, query}{total \, \# \, queries}$.

A correct query is one that is translated into a completely correct XQuery expression by NaLIX or DaNaLIX with semantic equivalent to the original natural language query (or other equivalent representations used by previous systems).

Training Data To make a direct comparison with previous systems, it is important to choose a training data set with a size similar to those used previously. COCKTAIL conducted ten-fold cross validation of the entire data set, while GENLEX made an explicit split between training and test data sets, with nearly 70% of **Geo880** and 80% of **Jobs640** queries used for training. Similar to GENLEX, we made an explicit split between training and test data sets: 85% of the queries in each data set were randomly chosen as training data, with the remaining queries as test data. It is important to realize that although our training data sets appear slightly larger than those used by GENLEX, the number of examples that can be utilized by our learning algorithm is actually smaller—a significant portion of queries in the training set do not contribute to new rules, since they can already be correctly translated by the system without domain knowledge. For example, the training data

in **Geo880** contains only 470 useful examples for DaNaLIX, whereas GENLEX used 600 training examples.

Procedure We first extracted a collection of rules from the training data sets of each domain. Next, for test data in each domain, we translated queries in the test data into XQuery expressions using DaNaLIX with an empty *knowledge base* (equivalent to running NaLIX). We then ran DaNaLIX again after loading the rules obtained from the training data. The XQuery expressions generated in each run were then recorded and manually examined for correctness. No ontology-based term expansion was used.

Results

Table 3 presents our experiment results. As can be seen, the learning and incorporation of domain knowledge in DaNaLIX has successfully improved recall over NaLIX on both datasets, with the recall increased from less than 70% to over 80%. In particular, the recall for queries on **Geo880** has been substantially increased by 129.8%. Such a remarkable improvement of DaNaLIX over NaLIX is not surprising—semantics of queries in **Geo880** often rely on terms with domain-specific meanings; a purely generic system like NaLIX simply cannot correctly interpret such terms. This result demonstrates that our approach can be especially useful in domains with rich domain-specific knowledge. For domains with less domain-specific knowledge, our approach still improves the system performance. For instance, jobs postings typically do not involve much domain knowledge. However, DaNaLIX still made an evident improvement of 19.63% on **Jobs640**.

Table 3 shows a slight decrease in precision in DaNaLIX. The small reduction in precision can mainly be attributed to two factors. First, both rule extraction and learning rely on dependency parse trees obtained from MINIPAR (Lin 1998), the dependency parser used by NaLIX. Like any other existing parser, MINIPAR may generate imperfect results,[6] leading to issues in both rule extraction and matching. Second, in our experiment, nodes in all the rules keep their default `matchCriteria` values assigned at rule creation time. The default `matchCriteria` values may cause a rule to be too general or too restrictive, resulting in errors in rule matching. In fact, in our experiment we found that a single rule extracted from the query pair for "*Find the states with the most cities*"— "$[id_0]$ (the most) $\Rightarrow$ **max**(**count**)($[id_0]$)"—has led to nearly 80% of the false positives in rule matching on **Geo880**. The default `matchCriteria` for $[id_0]$ is byType. As a result, this rule was wrongly matched with queries such as "*What is the cities with the most people?*" as both "*cities*" and "*people*" are of the same type (name token).

Table 3 also reports performance of GENLEX and COCKTAIL. When comparing the results, it is important to note that the comparison is not really about whether DaNaLIX can outperform the previous learning-based domain-specific systems, but to examine whether a generic system enhanced with domain-awareness using our approach can achieve performance similar to that of domain-specific systems. Since systems based on our approach are easily portable and require much lower manual effort in creating training data compared to previous domain-specific systems, we would be happy if DaNaLIX could achieve com-

---

[6]Minipar achieves about 88% precision and 80% recall with respect to dependency relations with the SUSANNE Corpus.

Table 3: Experimental results for NaLIX, DaNaLIX and previous systems.

	Geo880		Jobs640	
	P	R	P	R
NaLIX	97.87	35.07	100.00	68.75
DaNaLIX	89.85	80.60	98.71	81.25
GENLEX	96.25	79.29	97.36	79.29
COCKTAIL	89.92	79.40	93.25	79.84

P: Precision (%); R: Recall (%)

Source Tree		Target Tree
Visual Basic	$\Rightarrow$	VB
return (requirement (degree, of ([id_0])))	$\Rightarrow$	**return** (required degree (of ([id_0])))
(how long (**is**, [id_0]))	$\Rightarrow$	**return** (length (of ([id_0])))
[id_0] (**count** (people (in ([id_1]))))	$\Rightarrow$	[id_0] (population (of ([id_1])))
[id_0] (the longest)	$\Rightarrow$	[id_0] (with (**max** (length))
[id_0] (the most populous)	$\Rightarrow$	[id_0] (with (**max** (population))
[id_0] (bordering ([id_1]))	$\Rightarrow$	[id_0] (with (neighbor ([id_1]))
[id_0] (LISP Programmer ([id_1]))	$\Rightarrow$	[id_0] (Programmer (in (LISP ([id_1])))

Figure 5: Example Rules Learned In Our Experiment.
Symbol $\Rightarrow$ stands for "transform into."

parable performance. We were thrilled to find that DaNaLIX, a generic system with enhancement of domain knowledge learning and incorporation, actually achieves higher recall than GENLEX and COCKTAIL on both datasets and higher precision than both systems on **Jobs640**, with only a small reduction in precision on **Geo880**, mainly due to the same over-general rule noted earlier.

Figure 5 gives a sample list of domain knowledge rules that have been learned and applied by DaNaLIX in our experiment. We can see that these rules contribute important domain knowledge for the system to correctly interpret the corresponding queries. Furthermore, these rules are well generalized to be applicable to unseen queries.

Discussion
In our experiment, a query was sometimes matched with multiple rules. The existence of such conflicting rules was not accidental. It actually reflects several interesting issues worth further exploration. First, conflicting rules can result from ambiguous terms. For instance, we found more than one rule for the query "*Give me the largest state.*" One of the rules maps the query into "*Give me the state with the largest population,*" while another transforms it into "*Give me the state with the largest size.*" Both transformations may be considered correct, depending on the actual user needs. Second, conflicting rules may also be caused by the matchCriteria value of a node being overly general. For example, for the same query just noted, a matching rule obtained from "*What is the largest river?*" is found to match the query, and maps the query into "*Give me the state with the largest length,*" resulting in a query with incorrect semantics. Exploring techniques to address the above issues related to conflicting rules is an important area for future work. One interesting direction is to investigate how the matchCriteria for different nodes in each rule can be incrementally updated based on the statistics of the rule collection in the *knowledge base* to reduce the number of rules that are too general or too restrictive.

We have already mentioned that the imperfect results given by the parser may cause difficulties in both rule learning and matching. Even simple grammatical changes, such as adding or removing a modifier, may yield two dramatically different parse trees. Learning from such a pair of parse trees often results in a fairly narrow rule, which essentially requires the matching of the entire parse tree. Since such changes are commonly found in our training data, many rules learned are too narrow to be useful. In future work, we plan to investigate how to make our learning algorithm robust against such parser idiosyncrasies.

Conclusion
We have presented an approach to build a domain-aware natural language interface for querying databases with a generic framework. We extended the framework of an existing generic natural language interface for querying XML to allow domain knowledge extraction and incorporation. A key element of our approach is to automatically obtain domain knowledge from user interactions. This feature allows a generic NLIDB to directly take advantage of the user traffic—users of the same system can benefit from each other, by contributing to the knowledge base of the system while doing their normal work. Our experimental results demonstrate that such a system can improve search performance across different domains, with an especially significant advantage in domains with rich domain-specific knowledge such as geography, biology, and health care.

References
Androutsopoulos, I. *el al*. 1995. Natural language interfaces to databases — an introduction. *Journal of Language Engineering* 1(1):29–81.

Barendregt, H. 1984. *The lambda calculus, its syntax and semantics*. North-Holland.

Buitelaar, P. *el al*., ed. 2005. *Ontology Learning from Text: Methods, Evaluation and Applications*. Amsterdam, The Netherlands: IOS Press.

Gómez-pérez, A., and Manzano-macho, D. 2004. An overview of methods and tools for ontology learning from texts. *The Knowledge Engineering Review* 19:187–212.

Li, Y. *el al*. 2005. NaLIX: an Interactive Natural Language Interface for Querying XML. In *SIGMOD*, 900–902.

Li, Y. *el al*. 2006a. Constructing a Generic Natural Language Interface for an XML Database. In *EDBT*, 737–754.

Li, Y. *el al*. 2006b. Term Disambiguation in Natural Language Query for XML. In *FQAS*, 133–146.

Lin, D. 1998. Dependency-based evaluation of MINIPAR. In *Workshop on the Evaluation of Parsing Systems*.

Maedche, A., and Staab, S. 2001. Ontology learning for the semantic web. *IEEE Intelligent Systems* 16(2).

Popescu, A.-M. e. 2003. Towards a theory of natural language interfaces to databases. In *IUI*, 149–157.

Quirk, R. *el al*. 1985. *A Comprehensive Grammar of the English Language*. London: Longman.

Tang, L. R., and Mooney, R. J. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *ECML*, 149–157.

Xu, F. *el al*. 2002. A domain adaptive approach to automatic acquisition of domain relevant terms and their relations with bootstrapping. In *LREC*.

Zettlemoyer, L. S., and Collins, M. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, 658–666.