



Learning Rules for Conceptual Structure on the Web

HYOIL HAN
RAMEZ ELMASRI

hhan@cis.drexel.edu
elmasri@cse.uta.edu

College of Information Science and Technology, Drexel University; Department of Computer Science and Engineering, The University of Texas at Arlington

Abstract. This paper presents an infrastructure and methodology to extract conceptual structure from Web pages, which are mainly constructed by HTML tags and incomplete text. Human beings can easily read Web pages and grasp an idea about the conceptual structure of underlying data, but cannot handle excessive amounts of data due to lack of patience and time. However, it is extremely difficult for machines to accurately determine the content of Web pages due to lack of understanding of context and semantics. Our work provides a methodology and infrastructure to process Web data and extract the underlying conceptual structure, in particular relationships between ontological concepts using Inductive Logic Programming in order to help with automating the processing of the excessive amount of Web data by capturing its conceptual structures.

Keywords: ontology learning, information extraction, relational rule learning, knowledge discovery

1. Introduction

Web pages were originally designed to be human-readable and the best way to manage Web information is done by human beings. But manual work for Web information processing can be tedious, difficult, and time-consuming. Applying database techniques to the Web is one of the most promising approaches to manage Web data because of the possible applications of database management techniques (Florescu et al., 1998), which have been well developed over the last two decades. Data on the Web is growing tremendously and managing and querying this huge amount of data is extremely difficult, particularly because most Web data lacks explicit structure. Without conceptual structure for the underlying data, database techniques cannot be applied properly, because database query processing and management rely on the conceptual description, or schema, to represent the structure of underlying data.

Our work is a part of the WebOntEx (Web Ontology Extraction) project (Han, 2002), whose long-term goal is to automatically extract a Web ontology for a specific application domain. The WebOntEx system consists of three parts: Generic Pattern Extraction, Phrase/Synonym Extraction and Model Transformation. In this paper we present Generic Pattern Extraction process and the results. In WebOntEx, we hypothesize that there is an underlying conceptual structure to each Web page, even if the structure may be implicit. We consider Web page structure as part of the schema for an application domain. For the process of conceptual structure extraction, patterns for conceptual modeling of Web data that are in the same application domain are captured first. To discover structure, we use inductive logic programming (ILP) (Nienhuys-Cheng and Wold, 1997) to learn relevant rules to identify ontological concepts from implicit structure in Web pages.

The conceptual structure in our work is considered as a complete schema of the application domain concepts and is based on the Extended Entity-Relationship (EER) model (Elmasri and Navathe, 2000), which does not have axioms. Domain concepts are classified into entity types, relationships, and attributes and stored in a relational database to allow them to evolve over time. Our work can be considered as Web content mining according to the classification by Kosala and Blockeel (2000). Web content contains unstructured (that is, free text) and semi-structured data (that is, HTML (Raggett et al., 1999) and/or XML (Bray et al., 2000)). We capture patterns for conceptual structure on the Web data by analyzing Web content with HTML tag set and HTML tree structure and utilizing machine learning techniques and heuristic rules.

We do not know of any work directly related to the same goal as our work. Most other research work (Agichtein and Gravano, 2001; Brin, 1998; Califf, 1998; Craven et al., 1999; Craven and Slattery, 2001; Embley et al., 1998, 2001; Freitag, 1998; Maedche and Staab, 2000) extracts instances from the Web. On the other hand, we extract the patterns of type information for semantic relationships between ontological concepts from the Web. The rest of paper is organized as follows. In Section 2, generic pattern extraction process is explained. Section 3 describes the empirical results and analysis. Section 4 concludes our work and describes future research.

2. Pattern extraction

In this section, we explain the Generic Pattern Extraction (GPE) process whose aim is to extract patterns for relationships between ontological concepts of a specific application domain. GPE includes applying part-of-speech (POS) tagging and Extended Entity Relationship (EER) tagging to Web pages. The GPE process also conducts Feature Extraction (FE) on the annotated Web pages to discover the best feature, which is used as input to the Rule Learner that outputs Generic Patterns (GP).

2.1. Overall Generic Pattern Extraction process

Figure 1 shows the Generic Pattern Extraction (GPE) process for EER concepts from a collection of Web pages. The input of the process is a collection of Web pages from a domain and a set of domain-specific semantic concepts from a manually created ontology of the input application domain. The GPE process extracts patterns from annotated Web pages to discover the conceptual structure for unseen Web pages of the given domain. Current version of WebOntEx uses the extracted patterns to discover conceptual structure of the same domain. However the long-term goal of WebOntEx is to use the patterns to discover conceptual structure of unseen domains to capture a domain specific ontology.

The GPE process proceeds as follows: First, sentence-level part-of-speech (POS) tagging is conducted to distinguish noun, verb, adjective, and adverb. Then EER concepts for the input Web pages are annotated using Semantic Class, Synonym Set, and ontological concepts (see Section 2.2), which are manually crafted for the input domain to be used for input to the GPE process. The input Web pages belong to this pre-selected ontology. The Web pages annotated with POS and EER tags are used to create a Semantic Tree for feature

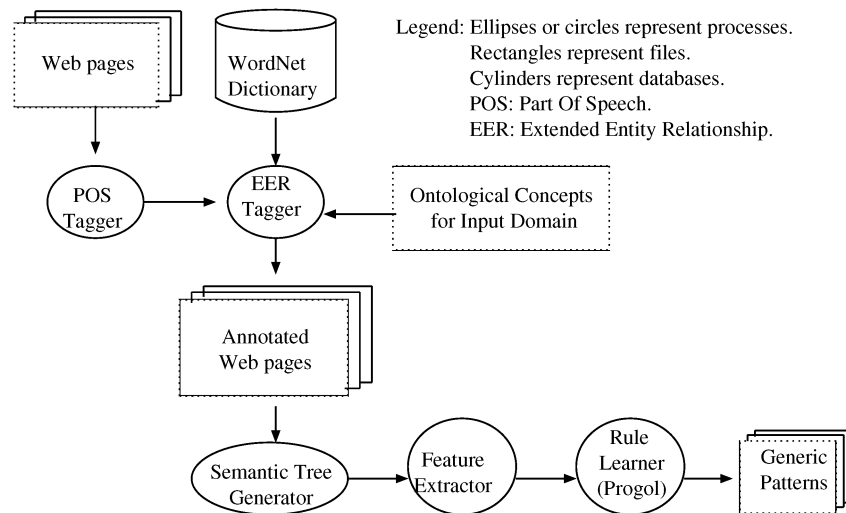


Figure 1. Generic Pattern Extraction process.

extraction (refer to Section 2.3). To extract patterns, we then use the publicly available Progol (Muggleton, 2001) system, which is an Inductive Logic Programming (ILP) system. To extract the relationships between ontological concepts in an application domain, we first identify patterns for relationships from the examples of ontological concepts on the Web pages of an application domain. The relations between examples can be expressed easily with first-order logic representations. Inductive learning of first-order rules is referred to as Inductive Logic Programming (Mitchell, 1997). Due to the power of this expressiveness, we use one of the Inductive Logic Programming (ILP) systems, Progol, which is downloadable from the Web site: <http://www.doc.ic.ac.uk/~shm/Software/progol4.5/>.

2.2. Annotating Web pages for concept learning

This section discusses how we annotate the HTML Web pages with POS tags (verb, noun, adverb, etc.) and EER concept tags (entity type, attribute, and relationship), and why they are necessary to extract semantic concepts from Web pages. The reason to annotate Web pages with POS and EER tags is to mark semantic concepts in order to create Semantic Trees from the annotated Web pages (see Section 2.3).

2.2.1. POS tagging. For POS tagging, we use a publicly available part-of-speech (POS) Tagger (Brill, 1995), which requires all input data be sentences, and is downloadable in <http://research.microsoft.com/~brill/>. We call it Brill's POS Tagger. To apply Brill's POS Tagger, all punctuation should first be separated from words (for example, "how are you?") should be changed to "how are you ?"). The Tagger was trained with articles from the Wall Street Journal, containing roughly 3 million words (Brill, 1995). The input to the

GPE (Generic Pattern Extraction) process consists of HTML Web pages, which generally do not satisfy the input condition of Brill's Tagger. To transform HTML Web pages into pure text sentences, we get rid of HTML tags on the Web pages and convert the broken phrases into sentences based on heuristics, which creates a sentence whenever it encounters punctuations. For the conversion, the HTML tag set is first divided into two parts: block-level tags and text-level (or inline) tags. Generally block-level tags may appear inside BODY in a HTML page, and contain other block-level tags and inline tags, whereas inline tags may contain only text or other inline tags. Block-level tags change the formatting of a block of text and cause a line break before and after the tag. On the other hand, inline tags change the formatting of a set of words within the text block and do not cause a line break.

The following preprocessing is required to process POS tagging:

1. HTML tags are divided into block-level tags and text-level tags based on HTML specification (Raggett et al., 1999) and heuristics.
2. HTML special characters of input Web pages are recognized, and converted to their display representation.

Table 1 shows some of the HTML tags from each tag set. Actually ANCHOR (<a>) is not a text-level tag in the original HTML specification (Raggett et al., 1999), but is categorized as an inline tag in GPE because it is not used to create a new line. The reason for distinguishing between block-level and text-level tags is to find sentences for input to Brill's POS Tagger (Brill, 1995) that works only with complete sentences. The block-level tags are used to identify paragraphs from which we identify sentences using heuristics. Our heuristics are the following: all text content between block-level tags is merged and creates a block paragraph. In this process HTML tags in the paragraph are removed to create a pure text paragraph, which is then divided into several sentences based on the existing punctuation in order to apply the Tagger, which works at the sentence level.

We then apply Brill's POS Tagger (Brill, 1995) for syntactic tagging to the sentences. The sentence-level syntactic tagging has higher accuracy compared to word-level syntactic tagging, because when each word is considered independently in word-level syntactic tagging, it is difficult to find its syntactic tag, since many words have multiple POS. After the POS tagging is completed, we recover the original HTML tags by comparing the original HTML Web page with the syntactically tagged text. The recovered Web page has HTML tags, and POS tags. Figure 2 shows the example of POS Tagging, where NNP, CD,

Table 1. HTML tagset separation.

Kind	Text level	Block level
HTML tag	font, tt, i, b, big, small, sub, sup, em, strong, code, samp, q, a, var dfn, var, cite, q, a	h1, h2, h3, h4, h5, h6, address, p, pre, div, center, blockquote, form, isindex, hr, table, caption, tr, th, td, ul, ol, li, dl, dt, dd, br, frame, frameset, body

Original HRML Web page:

```

<td><b>Instructor&nbsp;</b>
<br>Prof. John Smith
<br>CCB 138
<br>Phone: (404)894-2222
<br>Email: <a href="mailto:jsmith@cc.gatech.edu">jsmith@cc.gatech.edu</a>
<br>Office Hours: 11-12pm MWF</td>

```

The same Web page after POS tagging:

```

<td>
<b>Instructor/NNP</b>
<br>Prof/NNP ./ John/NNP Smith/NNP
<br>CCB/NNP 138/CD
<br>Phone/NN :/ (/ ( 404/CD )/SYM 894/CD -/: 2222/CD<br>
<a href="mailto:jsmith@cc.gatech.edu">Email/NNP :/: jsmith@cc.gatech.edu/JJ</a>
<br>Office/NN Hours/NNS :/: 11/CD -/: 12pm/CD MWF/NNP

```

Figure 2. An example of POS tagging.

and SYM represent POS tags. NNP means noun, CD means number, and SYM means symbol.

The POS tagging algorithm can be summed up as follows and is applied to input Web pages:

1. Merge all text content between block-level tags (START-START or START-END tags) to create a block paragraph. In this process, all HTML tags are removed.
2. Create sentences from paragraphs created in the above step in an input format compatible to the POS Tagger. Apply Brill's POS Tagger to the sentences created in the above step.
3. Recreate the same formatted Web page as the original HTML Web page content by combining the POS tagged sentences with the original HTML tags. The result is the HTML Web page with POS tags.

2.2.2. EER tagging. For each domain that is used for Generic Pattern Extraction (GPE), a set of ontological concepts for the application domains of the input Web pages are created manually, and stored in a table called *EER Concept Set (ECS)*, which also includes Synonym Sets and Semantic Classes. During this process, in addition to single word concepts, all possible bigrams used in the given domains are identified, and stored in ECS. The ECS contains most common concepts that we used to represent the conceptual structure of the given domain. The most common two word sequences in a text are normally referred to as *bigrams*. Bigrams simply represent common syntactic constructions involving individually extremely common words (Manning and Schutze, 1999).

For each concept e , Synonym Sets and Semantic Classes are created, where a *Synonym Set* $\{e\}$ contains synonymous words to a specific sense of an EER concept e , and a *Semantic Class* $[e]$ contains all kinds of abbreviations of an EER concept e . The $\{e\}$ and/or $[e]$ can be

empty. For example, the Synonym Set for a faculty could be {faculty, instructor, professor, teacher} and the Semantic Class for a teaching assistant could be [teaching assistant, TA, GTA]. The set of these concepts together is called the EER Concept Set (ECS), which includes morphologies (stem-words) only and is used for EER tagging. The ECS also contains the most commonly used bigrams in the given domain.

EER tags are annotated to a Web page after it already has its POS tags, which create an EER/POS Web page that has HTML tags, POS tags, and EER tags. In EER concept tagging, the GPE process uses WordNet to get the stem (or base) of each word according to its syntactic tag created by the POS tagging procedure, and annotates EER concepts by consulting ECS. When the EER concepts are annotated, preference is given to a bigram over a single word because the former is much more rare than the latter.

The EER tagging algorithm is applied to Web pages annotated with POS tags, and can be summed up as follows:

1. Stemming: WordNet is used to find the base word of a word based on its POS tag.
2. Consult EER Concept Set, which is created manually and includes domain specific ontological concepts, Semantic Classes and Synonym Sets to find EER concepts of the given domains.
3. Recognize EER concepts according to step 4.
4. Check bigram:
 - (A) Preference for first found bigram: The first found bigram has preference to a later found bigram. For example, once a word *A* builds a bigram with a word *B* in any two consecutive words *AB*, *B* is not considered to build a new bigram further, nor is *B* considered for a single concept.
 - (B) A bigram gets preference over recognizing a single word if and only if the consecutive words can build a bigram. If a word *A* does not build a bigram with a previous word, it is considered as a single independent concept until it is checked for the possible creation of bigram with the following word *B*. If *B* cannot build a bigram with *A*, finally *A* becomes a single word concept.
5. Attach a unique number to the recognized EER concept. This is used for identification of examples for input to the Rule Learner. (For example, entNo = 20062 in figures 5–7.)
6. Tag the recognized EER concepts with a unique number to identify each EER construct. (For example, E4 for Instructor in figures 5–7.)

Figure 3 shows the EER construct numbering of ontological concepts for a faculty domain. The EER constructs of figure 3 are manually created. The <#entNo=20062> represents a unique number to identify an entity type for the input to the Rule Learner. The final Web page has three types of tags: HTML tag, POS tag, and EER tag, and is called an EER/POS Web page, which is used for feature extraction. Figure 5 shows an example of an EER/POS Web page for a faculty domain. In figure 5, <EERTAG> represents the boundary of our own EER tag to annotate ontological concept in a specific domain. The <*E4> (or <*E4>) represents entity types and the number '4' means the fourth entity type among many entity types of a domain.

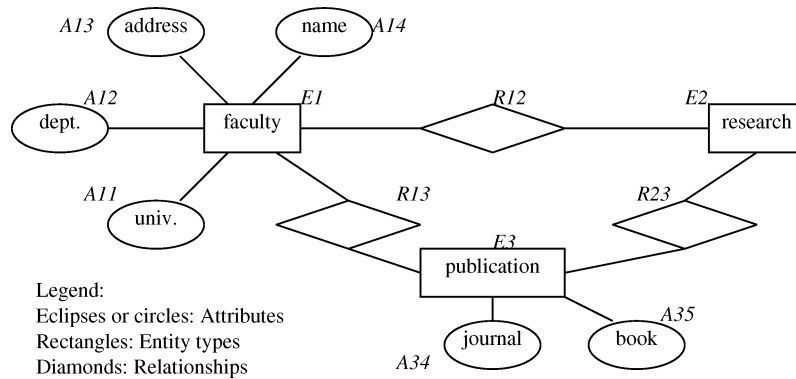


Figure 3. EER construct numbering.

```

<td><b>Instructor&nbsp;</b>
<br>Prof. John Smith
<br>CCB 138
<br>Phone: (404)894-2222
<br>Email: <a href="mailto:jsmith@cc.gatech.edu">jsmith@cc.gatech.edu</a>
<br>Office Hours: 11-12pm MWF</td>
    
```

Figure 4. A Chunk of a Web page.

```

<td>
<b><EERTAG><*<E4><#entNo=20062>Instructor/NNP<*<E4></EERTAG></b>
<br>Prof/NNP ./ John/NNP Smith/NNP
<br>CCB/NNP 138/CD
<br><EERTAG><*<A44><#attNo=2102>Phone/NN<*<A44></EERTAG> :/( ( 404/CD )/SYM
894/CD -/: 2222/CD<br>
<a href="mailto:jsmith@cc.gatech.edu">
<EERTAG><*<A45><#attNo=2103>Email/NNP<*<A45></EERTAG>
:/: jsmith@cc.gatech.edu/JJ</a>
<br><EERTAG><*<A43><#attNo=2105>Office/NN Hours/NNS<*<A43></EERTAG>
:/: 11/CD -/: 12pm/CD MWF/NNP
    
```

Figure 5. After EER tagging is applied to the Web page in figure 4.

2.3. Feature extraction

In this section, we describe the Feature Extraction (FE) process whose output is a feature set. The derived feature set is used for creating logic descriptions for input to the Progol (Muggleton, 2001) Rule Learner. The Progol input has two parts: background knowledge and positive/negative examples. FE provides logic descriptions for these two parts.

For Feature Extraction (FE), a Semantic Tree (ST) is created from each EER/POS Web page. Figure 5 shows an example of a Web page that is annotated with POS tags and EER tags. The Semantic Tree for the Web page in figure 5 is shown in figure 7, which is a part of 6. Figure 6 shows an example of a Semantic Tree and is used to describe the traversal and predicate creation. The Semantic Trees are traversed to extract the features describing

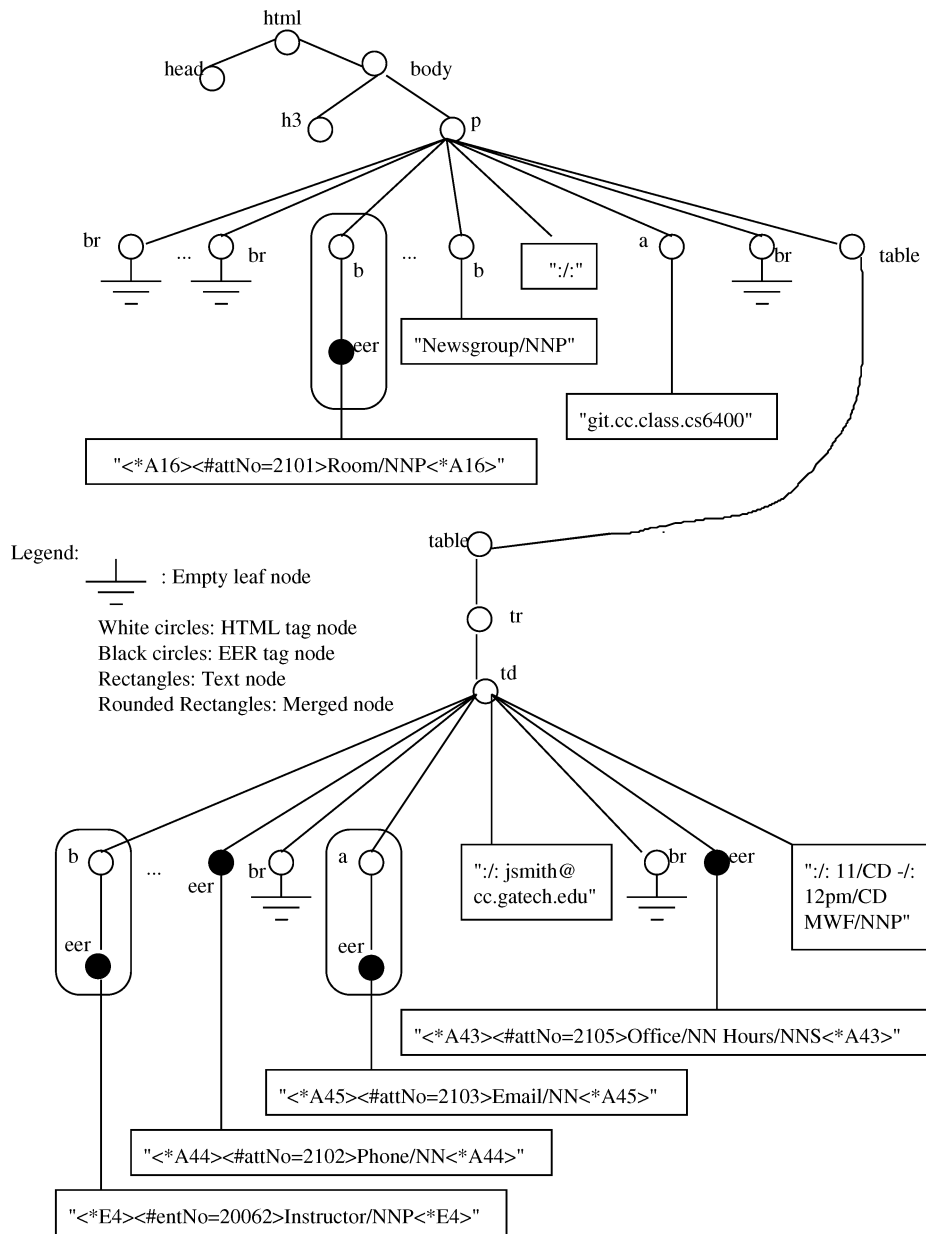


Figure 6. An example of a semantic tree for the Web page in faculty domain.

semantic concepts, and to create logic descriptions for an input to the Inductive Logic Programming (ILP) system to find Generic Patterns (GP). From the annotated Web pages with POS tag and EER concepts, we create first order logic descriptions to be used for an input to Progol, which will compute generic rule (or pattern) sets for EER concepts.

2.3.1. Semantic tree definition. The nodes of the Semantic Tree consist of EER tags, HTML tags, and text (the POS tags are considered to be part of text nodes). EER concepts are found at the child nodes of EER tag nodes of the Semantic Tree (ST), which is utilized for extracting features. ST is a rooted ordered tree T , which is a pair (N, E) , where N is a finite set and represents the node set of T , E is a binary relation on N and represents the edge set of T , and there is only one root r . The children of any nodes in T are ordered. The node set N consists of nodes N_{html} from HTML tags, nodes N_{eer} from EER tags, and nodes N_{text} from text. All internal nodes (or nonleaf nodes) of T consist of nodes from HTML tags, and nodes from EER tags. All leaf nodes of T consist of nodes from HTML tags and text nodes. For example, only white circles and black circles in figure 6 can be candidates for internal nodes of a semantic tree, where white circles represent HTML tag node and black circles represent EER Tag nodes.

Any EER tag node n_e (that is, $n_e \in N_{eer}$) has only one child node n_c , which is a text node and contains an EER concept of the given domain. This node n_c is called an EER Concept Node (ECN). Actually the text nodes N_{text} in T consist of the set N_c of ECN and the set N_t of pure text node n_t without EER concept, where $n_c \in N_c$ and $n_t \in N_t$, and n_t represents a pure text node without EER concept but with only text. For example, rectangular nodes in figure 6 represent text nodes, which consists of pure text nodes and EER concept nodes.

Following is a summary of the notation introduced in this paragraph:

$T = (N, E)$, where

$$N = N_{html} \cup N_{eer} \cup N_{text},$$

$$N_{text} = N_c \cup N_t, \text{ and } N_c \cap N_t = \emptyset.$$

N_{html} : the set of HTML tag nodes.

N_{eer} : the bag of EER tag nodes.

N_{text} : the set of text nodes.

N_c : the set of EER Concept nodes.

N_t : the set of text that does not contain terms (that is, EER concepts) of EER Concept Set (ECS), which contains all ontological concepts of the input domain.

For example, we can get the following node information from figure 7:

$$N_{html} = \{td, b, br, a\},$$

$$N_{eer} = \{eer, eer, eer, eer\},$$

$$N_{text} = \{\text{Instructor, Phone, Email, Office Hours, jsmith@cc.gatech.edu, 11-12pm MWF}\},$$

$$N_c = \{\text{Instructor, Phone, Email, Office Hours}\},$$

$$N_t = \{\text{jsmith@cc.gatech.edu, 11-12pm MWF}\}.$$

A path ($n_1 \sim n_2$) of a Semantic Tree from a node n_1 to a node n_2 is defined as a sequence of nodes from n_1 to n_2 including n_1 and n_2 . The nodes on this sequence should be in the same Semantic Tree under consideration. The length of the path is the number of edges between n_1 and n_2 . No edge should appear twice.

2.3.2. Semantic tree traversal and predicate creation. Semantic Tree traversal allows GPE process to collect features for input to the Rule Learner, and create logic descriptions, such as prepositional predicates and first order logic predicates. Whenever we encounter an EER tag node in a Semantic Tree traversal, we traverse back up to the root r of the ST (Semantic Tree) in order to retrieve all HTML tags in the path that has all its ancestors. A part of these HTML tags are used to create a set of features, which is one of the important factors in the GPE process to extract information from the Web. Among the HTML tags on the path, only tags in the *Default Tag Set (DTS)* whose members are `<title>`, `<head>`, `<h#>`, `<th>`, `<td>`, `<u>`, ``, ``, ``, `<i>`, ``, ``, `<object>`, `<input>`, `<caption>`, `<option>`, and `<a>`, were chosen based on an analysis of Web pages to determine which tags are most promising for the heuristic rules for ontology extraction. Among tags in DTS, tags that can be used to format layout similarly are merged together to decrease the number of features. The emphasized HTML tags consist of ``, ``, `<i>`, ``, and `<u>`. The listed form HTML tag consists of ``, and ``. All header level tags are merged as a header HTML tag. These merged tags appear in logic descriptions as *has_emphasized(eer)*, *has_listedForm(eer)*, and *has_header(eer)*, respectively, where *eer* represents an instance for an EER construct (such as an entity type or an attribute).

The HTML tags used to format an EER concept of n_c are all HTML tags in the path ($n_p \sim r$), where r is a root of the Semantic Tree and n_p is the parent of an EER tag node n_e , which is the parent node of an EER Concept Node n_c . For example, in figure 6, from *eer* node whose child node has concept '`<#attNo=2103>Email`', we create the following predicates based on HTML tags on the path from the *eer* node to the root: *has_td(2103)*, *has_tr(2103)*, and *has_p(2103)*.

All sibling nodes of the current EER tag node, say n_e , are also found. The GPE process uses two kinds of sibling nodes: (regular) Sibling node and Semantic Sibling node. A Sibling node is created based on the displayed HTML tree structure. If a sibling node, say n_s , of the current EER tag node n_e is an EER tag node, a logic description is created to describe relations between the current EER tag node n_e and the sibling node n_s if and only if n_s is also an EER tag node and its child node does not have the same EER concept as the child node of n_e under consideration. This n_s is called a (regular) Sibling node of n_e . For example, for an *eer* node whose child node is '`<#attNo=2102>Phone`', we create a predicate *hasSibling(2102, 2105)* with an *eer* node whose child node is '`<#attNo=2105>Office Hours`'. This is due to these two *eer* nodes being in the same level of a subtree.

In the case that the parent node n_p of the current EER tag node n_e has a text-level HTML tag (in Table 1), n_e is merged with its parent node to find the Semantic Sibling nodes of n_e (see the rounded rectangles in figure 6). That is, looking for the Semantic Sibling nodes of n_e is actually looking for the sibling nodes of its parent node in this case. This is because we try to find sibling structures of the HTML tree structure based on the HTML

block structure. This idea is mainly from the fact that building an HTML tree is based on block level HTML tags rather than text level HTML tags. This gives the clue about how the HTML block level tree structure affects the conceptual structure of the content of Web pages. For example, for an *eer* node whose child node is ‘<#entNo=20062>Instructor’, we create a predicate *hasSemanticSibling*(20062, 2102) with an *eer* node whose child node is ‘<#attNo=2102>Phone’. The reason is that these two *eer* nodes have one level difference of a subtree and the difference is caused by a text-level HTML tag, B, which does not affect the Semantic Tree structure.

Our Semantic Tree is ordered. A leftmost node is defined as a node that does not have a left Sibling node and is used for our feature selection. The leftmost node is meaningful because block-level HTML tags create a new subtree of our Semantic Tree and the first node of a subtree becomes a leftmost node.

In a Semantic Tree traversal, external links and internal links create feature sets for possible relationships between Web pages. We traverse a Semantic Tree from a root r by depth-first search.

The Semantic Tree Traversal algorithm can be summarized as follows: Whenever it meets an EER tag node n_e that has a child node of an EER concept c , conduct the following work. A first order logic predicate with a concept X and a concept Y , is created only if X is neither part of Synonym Set of Y nor part of Semantic Class of Y (see Section 2.2.2). This fact can be represented as $\exists e_i$ such that $c \notin \{e_i\}$ and $c \notin [e_i]$.

1. Traverse all ancestors up to the root r from the given EER tag node n_e to create a logic description with HTML features that are used to describe the current EER concept c from the current EER tag node to the root r .
2. Find all regular sibling nodes of the current EER tag node: for each sibling node n_s , create a logic description for a relation between c and the EER concept e_s in the sibling node n_s if and only if $\exists e_s$ such that $c \notin \{e_s\}$ and $c \notin [e_s]$.
3. Find all semantic sibling nodes of the current EER tag node if its parent node has a text-level HTML tag: for each sibling node n_s of its parent node create a logic description for a relation between e and the EER concept e_s in the sibling node n_s if and only if $\exists e_s$ such that $c \notin \{e_s\}$ and $c \notin [e_s]$.
4. Create logic descriptions for relations between the current EER concept c and its related EER concepts e_i existing in the same Web page, and both c and e_i exist in the same sub-tree. We create a logic description for a relation between c and e_i if and only if $\exists e_i$ such that $c \notin \{e_i\}$ and $c \notin [e_i]$. For example, *entityOf*(20062,2105) is created because both examples are in the same sub-tree and one (20062, Instructor) is an entity type of the other (2105, Office Hour) in figure 6.
5. Create a propositional logic description when the parent of n_e is the leftmost node among its siblings in a sub-tree.
6. Create relationships from link: When the traversal meets an ANCHOR (<a>), FRAME, or LINK tag with HREF or SRC attribute, create a logic description for a link relation between the current URL and the linked URL if and only if the attribute value of HREF or SRC is a real URL pointing to an HTML Web page with the same input organization under consideration (for example, if the input URL is from University of Washington, we accept links only from University of Washington) and within the given link depth.

This is to see how link relationships can be used to describe conceptual structure. (The link depth is given as an input. In our experiment, we gave the link depth as 1.)

The created logic descriptions are utilized to build training examples and test examples for input to Rule Learner, which induces rules (or patterns) from training examples with background knowledge, given as its input. We call the learned rules for positive examples Generic Patterns (GP), which are learned from input Web pages and describe the general usage of HTML tags and tree structure to describe EER concepts on the Web. The positive examples are described in the form of target relations, which are the predicates that we want to learn: $entityOf_{ij}(e_i, a_{ij})$, $hasRelationship_{ij}(e_i, e_j)$, and $superclassOf(e_i, e_j)$, where the e (or e_i) represents one of the entity types and the a_{ij} represents a j th attribute of an entity type e_i . The EER construct numbering for a faculty domain is shown in figure 3.

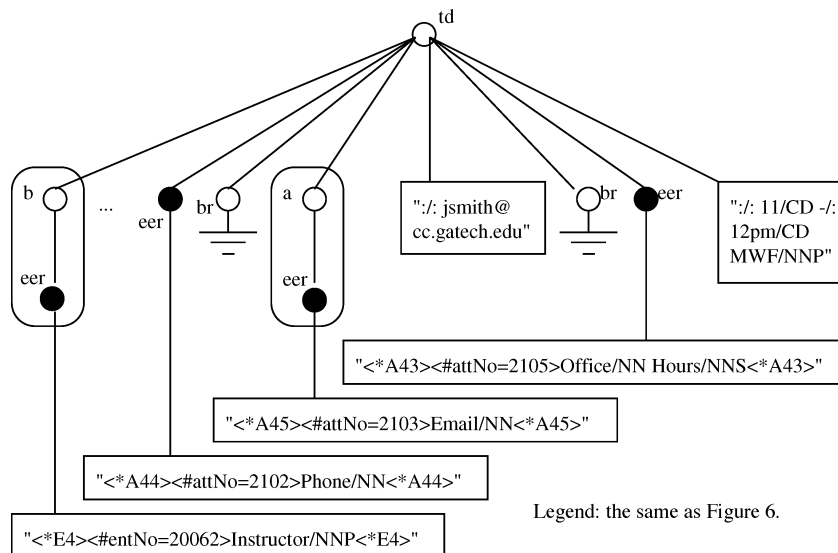
The background knowledge is from HTML tags in the Default Tag Set (DTS), HTML tree structure, and Web page link structure, and is described by two different representations, propositional logic and first-order logic. The following background knowledge is used for our experiments:

- A simple value-test predicate is represented by a propositional logic description: $has_title(eer)$, $has_emphasized(eer)$, $has_listedForm(eer)$, $has_p(eer)$, $has_a(eer)$, $has_dt(eer)$, $has_font(eer)$, $has_dd(eer)$, $leftmostNode(eer)$, where eer represents an instance for an EER construct (such as an entity type or an attribute). For example, $has_title(eer)$ means that an instance eer is depicted using the TITLE tag (one of HTML tags) in an input Web page.
- The relation between two input instances is represented by first-order logic descriptions $containedIn(eer, url)$, $connectedTo(url_i, url_j)$, $hasSemanticSibling(eer_i, eer_j)$, $hasSibling(eer_i, eer_j)$, where url represents a URL (Universal Resource Location), eer represents one of the EER constructs, and i is not equal to j . The $connectedTo$, $hasSemanticSibling$, and $hasSibling$ are commutative.

To show an example of logic descriptions (that is, predicates), an original Web page is shown in figure 4 and its annotated Web page is shown in figure 5. Its semantic tree is shown in figure 7. The logic descriptions for the annotated Web page in figure 5 are shown in figures 8 and 9. Figure 8 shows propositional predicates and figure 9 shows first order predicates (or relational predicates). These logic descriptions are used for the input to Progol.

2.4. Summary

In Section 2, we described the Generic Pattern Extraction (GPE) process that includes the annotation of POS and EER tags, and the feature extraction from a Semantic Tree, which is built on the annotated Web pages. The feature extraction process creates logic descriptions for the input to the Rule Learner (that is, Progol), which produces rules (that is, patterns). The results of running Progol are presented in Section 3.



containedIn(20062,gatechedu_11_0).	% *E4 Instructor/NNP
containedIn(2102,gatechedu_11_0).	% *A44 Phone/NN
containedIn(2103,gatechedu_11_0).	% *A45 Email/NNP
containedIn(2105,gatechedu_11_0).	% *A43 Office/NN Hours/NNS
entityOfA43(20062,2105).	% (*E4, *A43)
	% (Instructor/NNP, Office/NN Hours/NNS)
entityOfA44(20062,2102).	% (*E4, *A44)
	% (Instructor/NNP, Phone/NN)
entityOfA45(20062,2103).	% (*E4, *A45)
	% (Instructor/NNP, Email/NNP)
hasSemanticSibling(2102,20062).	% (*A44, *E4)
	% (Phone/NN, Instructor/NNP)
hasSemanticSibling(2103,20062).	% (*A45, *E4)
	% (Email/NNP, Instructor/NNP)
hasSemanticSibling(2105,2103).	% (*A43, *A45)
	% (Office/NN Hours/NNS, Email/NNP)
hasSemanticSibling(2105,20062).	% (*A43, *E4)
	% (Office/NN Hours/NNS, Instructor/NNP)
hasSibling(2102,2105).	% (*A43, *A44) (Office/NN Hours/NN, Phone/NN)

Figure 9. Relational predicates created from the Web page in figure 7.

contain more than 3 extraneous tags other than HTML tags (for example, XML pages would be discarded). For the faculty domain, we collected Web pages from University of Maryland (umd.edu). Among them, we selected 14 URLs from umd.edu after cleaning. For the CS department domain, we collected and selected 39 URLs from different universities after cleaning. When we extracted Web pages from those Root URLs, we also extracted Web pages linked to those Root URLs within the link depth 1. Therefore the number of Web pages extracted from those initial URLs varies. Because different Web sites use different design choices to build HTML Web pages (for example, how much information to place on the root Web page), the total number of annotated EER concepts per initial URL and their linked Web pages also varies.

To determine training and testing sets, we applied a Bootstrapping procedure (Liu and Motoda, 1998) that allows sampling with replacement because our data set is not large: it randomly picks N instances from the data set to form a training set, and the rest is used for a testing set (Liu and Motoda, 1998). In our experiments, we ran Progol per organization (that is, university) in a Web domain. When we determined training set and testing set per run, we randomly picked half of the Root URLs such that they provide both positive examples and negative examples for the target relation in the organization under consideration. For example, when we run Progol for Web pages of an organization, say umd.edu (total 14 URLs), we randomly picked half the URLs (7 URLs) for training data and the rest for testing data. We repeated this bootstrapping procedure 50 times, each time randomly choosing half the URLs for training and the remainder for testing. Although we picked half the URLs for the training set, the number of examples is not exactly half because the number of links per URL is varying, and the content of Web pages varies.

In our work, negative examples are determined by our own methodology. We created negative examples in the following way: when a target relation represents a relationship

Contingency table:

	A	~A	
P	a	b	(a+b)
~P	c	d	(c+d)
	(a+c)	(b+d)	

Figure 10. Contingency table.

between two entity types E_i and E_j , negative examples are created for all other relationships that exist in the ontology under consideration (for example, see figure 3 for a faculty domain) between two entity types E_m and E_n , where m and n are not i and j . In the case that a target relation is a relation $entityOf_{ij}$ between an entity type E_i and an attribute A_{ij} , the negative examples are created from all relations $entityOf_{ij}$ with entity type E_i and attribute $A_i y$, y is not j . When a target relation is $superclassOf_{ij}$, all the $superclassOf$ constructs other than $superclassOf_{ij}$ and all the $hasRelationship_{ij}$ relations existing in the ontology of the given domain are used for negative examples.

We determine a main entity type of the domain under consideration. Here, ‘main entity type’ means an entity type that is the most important in the domain. For example, ‘faculty’ is a main entity type of the ‘faculty’ domain, and ‘department’ for the ‘computer science department’ domain. We mainly conducted experiments for the relationship between the main entity type and other entity types, and between the main entity type and one of its attributes.

Progol outputs a contingency table that shows its accuracy and effectiveness. The contingency table shown in figure 10 (Parson and Muggleton, 1998) is interpreted in the following way:

The ‘P’ in figure 10 represents a correct prediction and ‘~P’ represents an incorrect prediction. The ‘A’ represents actual positive examples and ‘~A’ represents actual negative examples. Therefore ‘a’ means correctly predicted positive examples, ‘b’ means the number of examples that are actually negative examples but predicted as positive examples, and ‘c’ means the number of examples that are actually positive examples but predicted as negative examples. The *precision* P can be obtained by the formula $a/(a + b)$, and *recall* R can be calculated by the formula $a/(a + c)$.

The *F-measure* F can be calculated by the following formula, where β is a parameter between 1 and infinitely varying the degree of importance of precision P and recall R by weighting of precision P and recall R :

$$F_{\beta} = \frac{(\beta^2 + 1)PR}{(\beta)P + R}$$

In the next section, we show the results when β is 1, which gives equal weighting, to judge our result by a single value instead of two values: precision and recall. *F-measure*

gives information for the averaging of both precision and recall together. But precision and recall have different aims. In many research experiments, increasing one of the two usually results in decreasing the other. We concentrate on improving precision rather than maximizing recall because the Web has an abundance of information for extraction.

Progol also gives results using the well-known χ^2 (Chi-square) (Montgomery and Runger, 1994) test, giving a statistical measure of the significance of the results (Parson and Muggleton, 1998). In our experiments, we discarded the result if the associated χ^2 probability is greater than 0.05 (significance at the 5% level), because the result is not significant otherwise.

3.2. Results and analysis

In our relational learning (first-order predicate learning), we assume all of our annotated EER concepts are correct. The target relation that we run consists of three different kinds of relations: *entityOf*, *hasRelationship*, and *superclassOf*. Some examples of these rules are: *entityOf*(faculty, department), *entityOf*(faculty, university) and *hasRelationship*(faculty, publication). These are examples of the actual target relations.

In the faculty domain, the following are used as negative examples for *hasRelationship*(department, university): *hasRelationship*(department, course), *hasRelationship*(department, people), *hasRelationship*(department, announcement), *hasRelationship*(department, project). Notice that choosing these as negative examples does not mean that they are not correct, but only that they are used as negative examples for this particular target relationship.

For *superclassOf*(people, faculty), all relations of the same category of this *IS_A* construct are used as positive examples and all relations of other superclass/subclass constructs and relationships between entity types are used as negative examples. Positive examples consist of instances of *superclassOf*(people, staff), *superclassOf*(people, alumnus), *superclassOf*(people, student), and *superclassOf*(people, faculty). Negative examples consist of two parts: one from other *superclassOf* constructs in the same domain and others from any valid relationships between entity types such as *superclassOf*(student, graduate student), *superclassOf*(student, undergraduate student), *superclassOf*(announcement, award), *superclassOf*(announcement, event), *superclassOf*(announcement, colloquium), *hasRelationship*(department, university), *hasRelationship*(department, course), *hasRelationship*(department, people), *hasRelationship*(department, course), *hasRelationship*(department, announcement), and *hasRelationship*(department, project).

The experiment results are shown in Tables 2 and 3. The numbers in Tables 2 and 3 represents average values in percentage. The value in parentheses represents standard deviation. The number of runs that we experimented with is 50. Among them, only meaningful rules with Chi-square probability < 0.05 are collected and shown in Tables 2 and 3. After removing runs that produce trivial rules, the number of runs that were used to build the tables in Tables 2 and 3 is fewer than 50 in several cases and is shown in the rightmost columns of Tables 2 and 3.

We define trivial rules as the following: the most specific rules that cover only one example and the rules that have only one variable in the rule body. The latter is due to the property

Table 2. Results for computer science department domain.

EER construct	Precision	Recall	F1	No. of examples	No. of runs used
<i>hasRelationship</i> (dept., univ.)	87.78 (31.22)	17.80 (6.14)	14.08 (4.10)	90.93 (10.28)	14 (among 50)
<i>superclassOf</i> (people, faculty*)	74.28 (22.27)	21.84 (14.46)	15.02 (7.89)	150.67 (27.62)	6 (among 50)

Table 3. Results for faculty domain.

EER construct	Precision	Recall	F1	No. of examples	No. of runs used
<i>entityOf</i> (faculty, research)	76.83 (15.62)	21.64 (11.42)	16.10 (7.06)	447.50 (126.14)	24 (among 50)
<i>hasRelaionshiop</i> (research, publication)	62.06 (26.51)	54.42 (20.12)	25.63 (4.61)	658.38 (189.44)	16 (among 50)

of the target relation of the GPE process, which currently captures relationships between two EER constructs. Two variables in the body of a rule can give information for both EER constructs that constitute the target relation. For example, we consider that the following is a trivial rule because it does not provide the information of B in its body: $entityOf(A, B) :- has_listedForm(A), has_a(A)$.

The target relations that do not appear in Tables 2 and 3 have trivial rules after running Progol. In particular, the reason many target relations provide trivial rules in the CS department domain is that many CS department home pages use a lot of images and the GPE process is extracting patterns mainly from text on the Web. The asterisk in *superclassOf*(people, faculty*) of Table 2 represents that we learned rules for this target relation, and we created a semantic class {faculty, staff, student, alumni} to provide them as ‘a subclass’ corresponding to ‘a superclass’ people. With finer features than what the GPE process has, each concept in the semantic class could create patterns separately for the superclass/subclass relations.

The precision of relations in Tables 2 and 3 is higher than 60%, but the recall is very low. Therefore the F1 value is also very low. The meaning of low recall and high precision is that the learned patterns are working very well with certain cases, and not at all with other cases. This means that when we apply the learned patterns to extract an actual concept, we need to be selective and use some natural language processing techniques to help in clarifying the meaning of sentences or words that are extracted by the learned patterns. The high precision explains that the learned rules in the GPE process can capture accurate concept phrases from new Web pages. To overcome low recall, we merge the learned patterns from the Bootstrapping method to create a rule base that consists of the learned patterns. This would presumably increase recall slightly with a corresponding penalty in precision.

The examples of the learned rules (or Generic Patterns) that are created with precision/recall values in Tables 2 and 3 are as following:

1. hasRelationship(department, university):
 - hasRelationship(A,B) :- has_font(B), has_listedForm(A), has_p(B).
 - hasRelationship(A,B) :- has_font(A), has_p(B).
 - hasRelationship(A,B) :- has_p(B), leftmostNode(A).
2. superclass/subclass construct between people and faculty (or staff, student, alumni):
 - superclassOf(people, faculty):
 - superclassOf(A,B) :- has_emphasized(A), hasSemanticSibling(B, A).
 - superclassOf(A,B) :- has_a(A), has_a(B), has_font(A).
 - superclassOf(A,B) :- hasSemanticSibling(B,A).
 - superclassOf(A,B) :- has_a(A), has_a(B), has_emphasized(A).
 - superclassOf(A,B) :- has_a(A), has_a(B), has_emphasized(A), has_font(A).
3. entityOf(faculty, university):
 - entityOf(A,B) :- has_listedForm(A), hasSemanticSibling(B,A).
 - entityOf(A,B) :- has_emphasized(A), has_font(B).
 - entityOf(A,B) :- has_dd(A), has_font(B).
 - entityOf(A,B) :- has_dd(A), has_p(B).
 - entityOf(A,B) :- has_listedForm(A), has_p(B), leftmostNode(B).

These rules explain where the semantic relationships can be found on the Web. These rules (or patterns) are applied to new testing Web pages in the same domain and the results are shown in Tables 2 and 3.

In the CS department domain, we learned patterns for superclass/subclass (or *IS-A*) relationships, and this is very helpful for modeling data for databases. Actually extracting relationships is not an easy task. By improving the feature set and including some additional domain knowledge, we could improve the current precision and recall because we do not currently use any domain knowledge except for that provided through the EER tagging procedure.

4. Conclusions and future work

We explained an architecture and process of Generic Pattern Extraction (GPE), which annotates Web pages with POS tags and EER tags and creates predicates for the input to Rule Learner. We presented our promising results that explain the possibility of using HTML tags and tree structure to capture conceptual structure of Web pages. The precision of several semantic relationships for a given domain is larger than 62% (see Tables 2 and 3). In this work, the extracted rules did not include link relations, which could give inter-relationships of Web pages. The reason is that our work extracted data on the content on Web pages and found more intra-relationships of a Web page rather than inter-relationships of Web pages. Introducing more elaborate inter-relationships of Web pages could give more features for the learned rules.

The lack of semantic concepts on the Web could cause low performance of the GPE process. In a lot of cases, Web pages contain instance data rather than type information. For example, people write just name and title in their home pages such as “Joyce Smith, assistant professor,” rather than writing “my name is Joyce Smith and title is assistant professor.” In

this case, the GPE process should have a lifting algorithm or methodology to lift instances to type information. This is a very hard problem and other researchers do this manually. Also there are many words to express a concept for a specific domain. To find most of them is not easy either. EER Concept Set (ECS) that is the input to Generic Pattern Extraction (GPE) process should contain as many words representing correct concepts as possible and this will improve its performance. For example, in a Course domain, a lot of Web sites contain information about ‘classes’ or ‘schedules’ instead of ‘courses’. A more elaborated Semantic Classes and Synonym Set could improve the performance of GPE.

The extracted patterns for ontological relationships will be used for Phrase/Synonym Extraction in WebOntEx. In this paper, the extracted patterns are applied to new Web pages of the same domain. However in the future, it will be applied to new Web pages of other domains to find conceptual structure of other domains and shallow natural language processing will be applied to the concepts and relationships extracted by the patterns to capture the domain specific ontology. Currently we are working in this direction.

References

- Agichtein, E. and Gravano, L. (2001). Snowball: Extracting Relations from Large Plain-Text Collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*.
- Brill, E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*.
- Bray, T., Paoli, J., Sperberg-McQueen, C.M., and Maler, E. (2000). *Extensible Markup Language (XML) 1.0*, W3C Recommendation.
- Brin, S. (1998). Extracting Patterns and Relations from the World Wide Web. *ACM WebDB Workshop*.
- Califf, M.E. (1998). Relational Learning Techniques for Natural Language Information Extraction. PhD Thesis, The University of Texas at Austin, TX.
- Craven, M., Dipasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., and Slattery, S. (1999). Learning to Construct Knowledge Bases from the World Wide Web. *Artificial Intelligence*.
- Craven, M. and Slattery, S. (2001). Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *Machine Learning*, 43, 97–119.
- Elmasri, R., and Navathe, S.B. (2000). *Fundamentals of Database Systems*. Addison-Wesley.
- Embley, D.W., Kai, Y., and Xu, L. (2001). Recognizing Target-Ontology-Applicable Multiple-Record Web Documents. In *Proceedings of the 20th International Conference on Conceptual Modeling*.
- Embley, D.W., Campbell, D.M., Jiang, Y.S., Ng, Y., Smith, R.D., Liddle, S.W., and Quass, D.W. (1998). A Conceptual-Modeling Approach to Extracting Data from the Web. In *Proceedings of the 2nd International Conference on Conceptual Modeling*.
- Florescu, D., Levy, A., and Mendelzon, A. (1998). Database Techniques for the World Wide Web: A Survey. *ACM SIGMOD RECORD*, 27(3).
- Freitag, D. (1998). Machine Learning for Information Extraction in Informal Domains. PhD Thesis, Carnegie Mellon University, PA.
- Han, H. (2002). Conceptual Modeling and Ontology Extraction for Web Information. PhD Thesis, The University of Texas at Arlington, TX.
- Kosala, R. and Blockeel, H. (2000). Web Mining Research: A Survey. *SIGKDD Explorations*, 2, 1–15.
- Liu, H. and Motoda, H. (1998). *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers.
- Maedche, A. and Staab, S. (2000). Discovering Conceptual Relations from Text. *Proceedings of the 14th European Conference on Artificial Intelligence*, Amsterdam.
- Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press.
- Mitchell, T. (1997). *Machine Learning*. WCB/McGraw-Hill.

- Montgomery, D.C. and Runger, G.C. (1994). *Applied Statistics and Probability for Engineers*. John Wiley Sons, Inc.
- Muggleton, S. (2001). CProgol4.4: A Tutorial Introduction. In *Inductive Logic Programming and Knowledge Discovery in Databases*. Springer-Verlag.
- Nienhuys-Cheng, S. and Wold, R. (1997). *Foundations of Inductive Logic Programming*. Springer.
- Parson, R. and Muggleton, S. (1998). An Experiment with Browsers That Learn, *Machine Intelligence*, Vol. 15.
- Raggett, D., Hors, A.L., and Jacobs, I. (1999). *HTML 4.01 Specification*, W3C Recommendation.