Weaving Entities into Relations: From Page Retrieval to Relation Mining on the Web

Joseph M. Kelley, Kevin Chen-Chuan Chang, Tao Cheng, Shui-Lung Chuang, William Davis
Computer Science Department
University of Illinois at Urbana-Champaign
{jmkelley, kcchang, tcheng3, schuang2, wbdavis}@uiuc.edu

ABSTRACT

With its sheer amount of information, the Web is clearly an important frontier for data mining. While Web mining must start with content on the Web, there is no effective “search-based” mechanism to help sift through the information on the Web. Our goal is to provide a such online search-based facility for supporting query primitives, upon which Web mining applications can be built. As a first step, this paper aims at entity-relation discovery, or E-R discovery, as a useful function to weave scattered entities on the Web into coherent relations. To begin with, as our proposal, we formalize the concept of ER discovery. Further, to realize ER discovery, as our main thesis, we abstract tuple ranking—the essential challenge of ER discovery—as pattern-based cooccurrence analysis. Finally, as our key insight, we observe that such relation mining shares the same core functions as traditional page-retrieval systems, which enables us to build the new ER discovery upon today’s search engines, almost for free. We report our system prototype and tested, WISDM-ER, with real Web corpus. Our case studies have demonstrated a high promise, achieving 83% – 91% accuracy for real benchmark queries—and thus the real possibilities of enabling ad-hoc Web mining tasks with online ER discovery.

1. INTRODUCTION

There is nearly an endless wealth of information on the Internet. As an ultimate information source, with its sheer scale and wide diversity, the Web presents not only intriguing challenges for page retrieval but also promising opportunities for data mining. While Web content mining efforts (e.g., Yahoo or Google) have evolved in the past decade with significant efforts and advances for providing effective page retrieval. On the other hand, for exploiting the potential, the “mining” of Web content has however remained relatively unexplored.

In particular, as more Web content mining efforts (e.g., [6, 5, 14, 15]) have emerged, we observe a significant limitation (as Section 7 will further explain): While Web mining must start with content on the Web, there is no effective search-based mechanism to help sift through the information on the Web. That is, to leverage the full scale of the Web, mining techniques must be able to efficiently “search” interesting patterns by online query processing. To begin with, due to the lack of such mechanisms, many techniques simply crawl and scan Web pages, and thus do not scale well and are not suitable for ad-hoc tasks that must be processed online. Or, a small step further, other techniques rely on search engines as “pre-processing” to search for pages to analyze. While search engines are the most common (and probably the only) way to access Web data, their “keyword” queries are designed specifically for page retrieval.

Our goal is thus to provide a search-based facility for supporting query primitives, upon which Web mining applications can be built. Figure 1 illustrates our “ultimate goal” of WISDM—Web Indexing and Search for Data Mining—as a generic querying mechanism for facilitating a wide range of Web content mining tasks. We note that today’s search engines provide indexing and search facilities for retrieving of individual pages. Our goal of WISDM aims at building a layer of search functionalities that provide aggregate analysis of the Web holistically. To realize such a Web “mining” engine, we must ask: What are the useful functions to provide? How to abstract them as query primitives? How to support such primitives as online query processing?

As a step toward this goal, to address the first question, this paper aims at supporting entity-relation discovery, or E-R discovery, as a useful function. In essence, ER discovery is to associate named entities (e.g., prof for professor names, univ for universities) as individual pieces of information into a relation as a connected whole (e.g., (David DeWitt, Univ. Wisconsin)). Our goal is to provide systematic support for discovering a target relation whose “schema” consists of certain entities. Figure 2 conceptually illustrates this ER discovery— which we call the WISDM-ER system: For instance, if we are to find a relation with three entities (prof, phone, email) as its schema, WISDM may return[1] the relation R1. In particular, such ER discovery returns a set of tuples, each of which associates entities of certain types (as the schema specified)— e.g., the first tuple of R1 connects three entities: prof = David DeWitt, phone = 608-262-1204, and

[1]To give a practical context of our discussion, this paper uses real examples as discovered by WISDM.
As Section 2 will formalize the concept, we believe ER discovery a useful query primitive for building Web mining applications—To begin with, such discovery finds not only entities but also their “meaningful” associations, which is often the very essence of our quest of information (or even “knowledge”). For instance, by discovering the example relations $R_1$ and $R_2$, we will be able to build a few interesting applications: To establish a practical context, we will use both CSContact and CSResearch below as our motivating examples and “benchmark” scenarios.

- **CSContact**: By weaving entities $\text{prof}$, $\text{phone}$, $\text{email}$ into a relation $(\text{prof}, \text{phone}, \text{email})$, we can ask: What is the phone and email of, say, David DeWitt? What are the email of all prof at Wisconsin? There are many other questions we can answer.

- **CSResearch**: By weaving entities $\text{prof}$, $\text{univ}$, $\text{research}$ into a relation $(\text{prof}, \text{univ}, \text{research})$, we can ask: What is the research area of DeWitt? Who are database professors at various universities? Which area has the most faculty at Wisconsin?

Further, such discovery gives structure to entities on the Web, by linking them into relations, which thus opens up advanced database-oriented processing. Useful relations can be periodically discovered from the Web, stored in databases, and queried with other structured information already available—e.g.:

- By joining $R_1$ with $R_2$: What are the emails of the database professors at Wisconsin?
- By joining $R_2$ with a “university ranking” database: Which top-20 university has the most database faculty?

To enable such discovery, as our second question, what query primitives to support? At the core of ER discovery, our main challenge is to find promising tuples— or semantically meaningful associations of entities. As our main thesis, we propose to abstract this core task as pattern-based cooccurrence analysis. Note that our challenge is to “weave” entities into relations—We observe that such associated entities often materialize themselves as cooccurred patterns in Web text. Thus, we propose to holistically analyze many Web pages to associate entity terms that cooccur frequently in certain patterns. To motivate this abstraction, as our foundation, Section 3 develops dual hypotheses on how desired “tuple semantics” presents itself on the Web with holistic regularities.

Finally, to address our third question— How to realize ER discovery with online query processing? We build our solutions upon current search engines—As our key insight, we observe that while a traditional search engine (Figure 3a) indexes only “keywords” and returns only “pages,” at its heart, it essentially share the same core functions of ER discovery.

This very insight enables us to build our ER discovery almost for free—by turning a page-retrieval engine into a relation-mining system, as Figure 3 contrasts. At the input, we extend a keyword-only system to be entity-aware by extracting entities from Web pages. At the output, we morph a page-retrieval system to perform relation-discovery by ranking tuples with cooccurrence analysis and constructing relations accordingly. At the heart, our ER discovery share the same core functions—term indexing and pattern matching. With this insight, on one hand, we can now easily deploy such ER discovery on today’s search engines almost for free; on the other hand, our relation mining can coexist with page retrieval, providing a likely synergistic combination. Section 4 presents our “morphing” from page retrieval to relation mining, and Section 5 our concrete query primitives for realizing cooccurrence analysis.

Toward building WISDM-ER, we have developed a functioning prototype, upon the Lemur text engine [1]. With CSContact and CSResearch as our driving “benchmark” applications, our testbed crawls and indexes six computer science departments, with 82453 web pages and 1.4GB of raw text. Section 6 reports our system implementation efforts and case studies. Our studies have revealed the high promise of large scale ER discovery— In many cases, our “benchmark” queries achieve 83% — 91% accuracy for constructing complex relations. For further reference, our system demo is available online at wisdm-er.myftp.org.

We summarize the main contributions of this paper:

1. We propose the concept of entity-relation discovery as a useful function for a Web mining platform.
2. We abstract ER discovery as pattern-based cooccurrence analysis with a suite of query operators as its realization.
3. We build online query processing upon current text search engines, and thus extend Web page retrieval to relation mining almost for free.
4. We develop a prototype tested with real Web corpus, and demonstrate two case studies of Web mining applications.

2. ER DISCOVERY:

**WEAVING ENTITIES INTO RELATIONS**

Our goal is to provide ER discovery as a basic concept for Web mining. This section motivates and formalizes the abstraction of this concept upon which we will start its realization in Section 3.

To begin with, we note that our information quest is often to find certain “fact.” We take a view that a desired fact is essentially a tuple— or an association of entities, which forms a tuple. An entity (or called “named entity” in information extraction) is a domain of literal values—e.g., prof as a set of professor names, univ for universities, and email for email addresses. We may ask— What is the email of prof DeWitt? What are the univ of various prof? Finding the “desired” association (Section 3 will discuss such “tuple semantics”) is thus a concrete task for many Web mining applications.

We thus propose entity-relation discovery: Upon $W$ as a corpus of Web pages, ER discovery constructs a “target relation,” by
The goal of entity extraction is thus to construct the entity set $E$. 

Let us assume that the application has provided a request function $r$ that acts as input to the extraction process. The request function $r$ takes as input a set of entities and returns a set of tuples.

The extraction process consists of the following steps:

1. **Entity Recognition**: The application provides a set of entities $E$ which need to be extracted from the text.

2. **Tuple Ranking**: Given the set of entities $E$, the extraction function $r$ assigns a score to each tuple.

3. **Tuples Selection**: The tuples with the highest scores are selected from the set of tuples.

4. **Tuple Construction**: The selected tuples are organized into a target relation.

5. **Relation Construction**: The target relation is constructed by linking the selected tuples.

6. **Relation Evaluation**: The constructed relation is evaluated for its effectiveness.

Example 2 (Tuple Ranking): Consider the example where the application provides the set of entities $E = \{E_1, E_2, E_3\}$. The extraction function $r$ assigns the following scores to the tuples:

- $r(T_1) = 0.8$,
- $r(T_2) = 0.7$,
- $r(T_3) = 0.9$.

The tuples $T_3$ and $T_1$ are selected for further processing.

Example 3 (Relation Construction): Consider the example where the selected tuples are:

- $T_1 = \langle E_1, E_2, E_3, \ldots, E_n \rangle$,
- $T_2 = \langle E_1, E_2, E_3, \ldots, E_n \rangle$.

The construction function $c$ is applied to these tuples to construct a relation:

$\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$.
such a function for how a tuple forms the relationship of interest (as Section 3 will discuss). As Figure 4 shows, effectively, this step “transforms” the unordered cartesian product \( U \) into a ranked list \( \hat{U}_F \) as \((t_1, t_2, t_3, \ldots)\), with scores \( F(t_1) = .95, F(t_2) = .85\), and so on. If the function is effective, the ranking would “surface” those meaningful tuples to the top—e.g., \( t_1 \) represents a correct tuple (with respect to the interest of CSResearch).

### 2.1.3 Relation Construction

Upon the ranked universe of tuples, ER discovery will finally construct a relation as the output. What tuples should be included? Although tuple ranking has surfaced meaningful entity associations, not all such top tuples should be returned. To begin with, as tuples are ranked, we may only want a few top-\( k \) answers. For (prof, phone, email), we may pick one tuple per prof, as her major contact. For (prof, univ, research), each prof should associate with one univ, which may together associate with multiple research (e.g., in Figure 4, tuple \( t_1 \) and \( t_3 \) on \( H \) \( \vee \) Jagadish agree on univ but differ in research).

As the final step, relation construction assemble promising tuples into a target relation, satisfying some global relation constraints \( C \) as another objective parameter. Note that, to contrast, while tuple ranking focuses on “locally” associating entities into potential tuples as guided by \( F \), this step “globally” selects these tuples to construct a relation conforming to \( C \). Like \( F \), such relation constraints again depend on application semantics—for defining a “meaningful” relation for the application at hands.

In principle, relation constraints \( C \) can be any (one or multiple) criteria for a relation to satisfy—In particular, while ER discovery is a new task, many “traditional” constraints from relational DBMS are applicable, which we enumerate a few below:

- **Relation cardinality**: How many tuples to return? That is, as tuples are ranked, what top-\( k \) results?
- **Key constraint**: Is certain entity (i.e., attribute) \( E_i \) necessarily unique? That is, is \( E_i \) a “key” constraint? For instance, as just mentioned, for (prof, phone, email), we may have: prof \( \rightarrow \) phone email, i.e., prof is a key.
- **Functional dependencies**: Are there certain dependencies between entities \( E_i \) and \( E_j \)? For instance, for (prof, univ, research), as just mentioned, prof \( \rightarrow \) univ (although prof \( \not\rightarrow \) research).
- **Referential integrity**: Although each entity \( E_i \) can in principle take any instance from \( E_i(W) \), are their any restrictions to “reference” only a subset of the domain?—An application may be interested in only some specific instances. For our example of (prof, univ, research), we may restrict univ to, say, only those universities in California, i.e., univ may only reference \{Stanford, Berkeley,\}.

**Example 3 (Relation Construction)**: Suppose CSResearch specifies \( C \) as: [prof is a key]. Figure 4 shows the result relation, which enforces one tuple per prof. In implementation, to fulfill this \( C \), in relation construction, ER discovery can start from the top of \( \hat{U}_F \) (as the result of tuple ranking), select the first tuple of every prof, and construct \( R = \{t_1, t_2, t_3\} \). Thus, \( t_3 \) is not included, as it “duplicates” \( t_3 \) in terms of prof, violating \( C \).

Note that such relation constraints depend on the application objectives. To contrast, a different application may instead specify that it wants to return 3 research per prof (so prof is not a key)—e.g., to post-process the result relation to pick the best matching research for each prof (as ER discovery may not always find the correct tuple at the top), or simply to capture that a prof can naturally have multiple research areas.

### 2.2 ER Discovery

As we have motivated, we propose ER discovery as a task of weaving entities into relations: We abstract this task with three objective parameters \( (S, F, C) \), which respectively guide the three progressively larger units of construction—entities, tuples, and relations: Starting from the entities specified in the target schema \( S \), we associate their instances to identify promising tuples by tuple function \( F \), from which we construct a target relation that satisfies constraints \( C \). More formally, we define ER discovery as follows:

**Definition 1 (Entity-Relation Discovery)**: Let \( E_1, \ldots, E_n \) each be an entity, whose domain over \( W \), or \( E_i(W) \), is the instances of \( E_i \) occurring in \( W \). An ER discovery task \( Q = (S, F, C) \), given a schema \( S = \langle E_1, \ldots, E_n \rangle \), a tuple function \( F \), and a set of relation constraints \( C \), is to find a target relation \( R \), such that

1. \( R \subseteq U \), where \( U = E_1(W) \times \cdots \times E_n(W) \) is the tuple universe as \( S \) specifies,
2. each tuple \( t \in R \) is ranked sufficiently high by \( F \), and
3. the relation \( R \) satisfies \( C \).

We note that Definition 1 intends to generally capture the concept of ER discovery—without specifying implementation details such as what “sufficiently high” may actually translated to. Our view is that, while implementations may differ, ER discovery is a general Web mining concept for constructing coherent structure (the target relation) from the unstructured Web, and thus a useful query primitive for building Web mining applications upon (e.g., CSContact and CSResearch).

To conclude our abstraction of defining ER discovery in principle and to start our specific implementation in practice, we present the “system query interface” of our WISDM-ER. Figure 5 shows the interface, which consists of three groups (annotated S&F, C, and P). To illustrate, we fill our CSResearch example query (as Figure 4 overviews), but with restriction of university to U. Wisconsin. Thus, we are only interested in (prof, univ, research) where \( \text{univ} \subseteq \{ \text{U Wisconsin} \} \). (As Section 6 will report, this setting is benchmark query \( R4 \) in our actual case studies; see Figure 13.)

Our realization of ER discovery specifies a task \( Q = (S, F, C) \) as follows:

In Group “S&F”, the input field Tuple Function specifies \( F \), e.g., as Figure 5 shows, \( F = \#\text{dist-uw100}(\ldots) \). (We will explain the \( F \)-related constructs in Section 5.) To simplify the interface, this input also implies a schema as those entities appearing in \( F \), e.g., \( S = \langle \text{professor, university, phone} \rangle \).

Group “C” then specifies relation constraints \( C \): In particular, we implement 1) relation cardinality: by specifying # Tuples, 2) key constraint: by specifying Unique On, and 3) referential integrity: by specifying Reference Only. Thus, overall, for the filled query, we have \( C = \langle \text{professor is a key; university} \supseteq \{ \text{U Wisconsin} \} \rangle \).

Finally, Group “P” is an added feature, for specifying the output presentation. Links Per Tuple requests the number of Web pages to return as “evidences” for each tuple. (As Section 5 will discuss, as a result of cooccurrence analysis, each tuple will have a set of “supporting pages” in which the pattern occurs.) In addition, Order By specifies the order of listing tuples (e.g., by research alphabetically)—much like the same clause in SQL.

With our general proposal (Definition 1) and specific adaptation (Figure 5) of ER discovery, we are next to bring forward its realization. As our discussion has suggested, the core challenge lies in the second conceptual step—tuple ranking. We will start this realization with developing our key insight—tuple ranking as holistic cooccurrence analysis.

### 3. MOTIVATION: COOCCURRENCE ANALYSIS
As our task of tuple ranking, as a core step in ER discovery, is to identify the “meaningfulness” of entity associations—which we call tuple semantics, is not only implicit but also depends on specific applications. As such, the discovery of Web entity relationships (ER) is a challenging task. To begin with, each ER involves a tuple, which can be defined as a set of cooccurring terms in Web pages. For example, in the context of the White House, we can observe the cooccurrence of the terms "White House," "The White House," and "The U.S. Presidential Mansion." These terms can be used to identify a tuple, such as ⟨White House, The White House, The U.S. Presidential Mansion⟩. Our task is to find such meaningful tuples from the potential associations of entities on Web pages.

To capture this insight, we thus make two hypotheses: (1) discovery methods for finding tuples from the potential associations of entities on Web pages, and (2) patterns of cooccurrence analysis, which the hypothesis can guide us to identify. As examples, Figure 7 shows several Web page snippets that may reflect such patterns. For instance, in the snippet "The White House is located at 1600 Pennsylvania Avenue in Washington, DC," we can observe the cooccurrence of the terms "White House," "1600 Pennsylvania Avenue," and "Washington, DC." These terms can be used to identify a tuple, such as ⟨White House, 1600 Pennsylvania Avenue, Washington, DC⟩. Our hypothesis is that such patterns exist, and we can observe them across a large scale if such patterns exist, can we observe them across a large scale? If so, do we have any patterns that reflect some deep underlying Web pages, as our hypothesis suggests? To answer these questions, we explore the large scale of the Web to perform ER discovery.

As further evidence, we stress that several earlier works have observed these patterns. In particular, several recent Web "question answering" techniques have been developed to address the challenge of answering questions on the Web. For example, the system of the White House asks, "Where is the White House?" and the Web pages search for the White House's location. This question can be considered a query, and the answer to this question can be considered the tuple semantics of the query. As the basis of our approach, we propose to realize tuple ranking by pattern-based cooccurrence analysis, which can guide our tuple discovery, by exploiting association rules and other techniques. We explore the large scale of the Web to associate entity instances cooccurring in Web pages in a particular order and to discover tuple semantics, which can be used to reveal the "holistic" hidden regularity of Web pages.
converge at a large scale. To begin with, since the Web has become our “ultimate information source,” the need of Web *usability* naturally pushes common design patterns that many pages will likely follow. Further, as the Web is a heavily interlinked community, as in any social networks, “peer influence” will naturally forge the convergence of conventions. For instance, most personal homepages follow similar format or adopt uniform templates. In fact, there may even converge to de facto standards– e.g., online bookstores seem to follow Amazon.com as a standard interface.

As further evidences, several earlier works have explored cooccurrence analysis, at a large scale, albeit in an ad hoc way, for their specific mining tasks. Such analysis essentially features frequency counting over a large collection. For instance, WSQ/DSQ proposes *Web supported queries* which essentially count the frequency of term cooccurrences (e.g., how terms “sigmod” and “databases” cooccur), and use a search engine to execute the frequency counting (e.g., by keyword query “sigmod databases”). As a more classic example, for mining structured market-basket data (instead of the unstructured Web), mining association rules [3] boils down to counting the frequency of cooccurrence in transactions. As Section 7 contrasted, we believe these early explorations have helped pave the way for our hypotheses– Building upon the same insight, we aim at formally abstracting such analysis for enabling ER discovery (Section 2) and provide systematic search-based support (Section 4).

**Putting Together:** As we have argued, we believe that the dual hypotheses are not only currently observable– the nature of the Web (which facilitates pattern emergence and convergence) will continue to uphold their relevance. Putting together, we thus propose pattern-based cooccurrence analysis as a promising approach for tuple ranking– as the inverse discovery of Figure 6.

As motivated above, there are two key tasks for this discovery: Pattern matching and cooccurrence analysis. We thus develop our tuple ranking construct– i.e., the tuple function $F$ (Figure 4)– to essentially consist of a pattern (for matching patterns individually) and a *scoring* function (for scoring matched cooccurrences holistically). To concretely abstract tuple ranking as pattern-based cooccurrence analysis, Section 5 will develop a suite of “query primitives.”

Finally, we stress that, by this cooccurrence abstraction of tuple ranking– the core of our ER discovery– we can build our new relation-mining system on the same core pattern-matching engine of a traditional page-retrieval system. That is, we can realize our ER discovery almost for free– which Section 4 will develop next.

### 4. SYSTEM ARCHITECTURE:

**FROM PAGE RETRIEVAL TO RELATION MINING**

Our goal is to extend a traditional page retrieval system into an ER discovery system. This section describes the key insights and the architecture that allows this extension almost for free. In addition, through a description of the extensions, we will characterize what we mean by almost free.

First, a key insight is that both systems are essentially cooccurrence and pattern matching systems for the Web, as shown Figure 3. Specifically, the Web Crawler and Inverted-List Indexer, which collect Web pages and index the terms offline, are means to support efficient online queries. The core task of an online query is to search for a user-specified pattern, which equates to cooccurrence between the search terms in the user query. This cooccurrence and pattern matching task is handled by the Pattern Matcher.

Although these core tasks are shared between the two systems, there are two key differences. *First*, our ER discovery system is designed to handle *abstract entities*, in addition to concrete keywords. To support this, we have implemented an Entity Extractor that will recognize abstract entities, and we have generalized the Inverted-List Indexer and Pattern Matcher to handle these entities. *Second*, the end goal of the two systems is different. In the case of a page retrieval system, the end goal is to return Web pages, while the end goal of an ER discovery system is to provide relations between instances of entities. Thus, we have replaced the Page Ranker with the Tuple Ranker and Relation Constructor, in order to discover relations. This section proceeds by stepping through an example that describes the key similarities and differences in more detail.

**Example 4 (Page Retrieval vs. ER Discovery):** Figure 4 illustrates the process that a traditional page retrieval system and our ER discovery system undertake for similar queries, where both the documents and queries deal with professors and universities.

Specifically, Figure 4 shows the Web documents on the left, the process of a page retrieval system on top, and the process of our ER discovery system on the bottom. The query for the page retrieval system, #uw50(dewitt university), is searching for documents with the keywords *dewitt* and *university* within 50 words of each other, as specified by the #uw50. The tuple function for the ER discovery system is #tf-uw50(#entity(prof)#entity(univ)), where the ER discovery system will discover relations between prof and univ entities by finding instances of the entities that appear within 50 words of each other. In addition, #tf-uw50 specifies the tuple scoring function, which will be discussed more in Section 4.2.

From Figure 4, both systems are provided the same three documents, however, as an end result, the page retrieval system returns Web documents and the ER discovery system returns discovered relations. In the following sections we will step through Figure 4 to show how we can extend the architecture of a page retrieval system to discover these relations, almost for free.

#### 4.1 Basis: Traditional Search Engine

This section describes the architecture of a page retrieval system, which acts as the basis of our ER discovery system. We will step through Figure 4 to clearly show the execution of a page retrieval system. Then, Section 4.2 will describe the process of our ER discovery system, which will demonstrate how we extend the base architecture of a page retrieval system.

As shown in Figure 4, the page retrieval system begins with snippets from the home pages of David DeWitt and H V Jagadish, represented by the documents $D_1$, $D_2$, and $D_3$. This corpus is collected by a Web Crawler, as shown in Figure 3.

Next, the Inverted-List Indexer constructs an inverted index, which, given a keyword, $K_i$, returns a document-position list, or DP list. The document-position list stores the Web document IDs and the word positions for every appearance of $K_i$. For example, Figure 4 shows that a lookup on *dewitt* returns a DP list with three entries, each specifying the document ID and position, where the position is represented as a range.

The crawling and indexing occur offline in order to support efficient online queries. As previously mentioned, we will consider the query #uw50(*dewitt university*). Therefore, the Pattern Matcher utilizes the inverted index to find the matched patterns. This is accomplished by first performing inverted index lookups on *dewitt* and *university*. Then, the Pattern Matcher enforces the cooccurrence constraint, in this case, the positions of the words must be within 50. Finally, the Pattern Matcher constructs the *matchings table*, as shown in Figure 4. The matchings tables store the documents and position ranges of the matched patterns.

During the final stage, the Page Ranker applies a scoring measure on the matching tables, such as PageRank or a distance measure, to
As we discussed, the first key difference between a traditional search engine and our ER discovery system is that our ER discovery system searches over abstract entities, rather than just concrete values. As with the Inverted-List Indexer, the Pattern Matcher must be generalized to handle abstract entities. From Fig. 7, we can see that the online tuple function will consider the concrete instances values, which form tuples, such as ⟨prof, email⟩, rather than just the concrete values such as David DeWitt.<u>uw</u>. As with the Inverted-List Indexer, the concrete instance values must also be stored. Therefore, we must identify entities within a corpus of Web documents. Specifically, the Entity Extractor identifies and labels certain entities within the text. There is much research in this area. In this example, the Inverted-List Indexer builds an inverted index of entities, resulting in augmented documents D1*, D2*, and D3*. Entities are identified within documents from Figure 4.

In the final stage, the Relation Constructor applies the relation multiplication will have specific knowledge about its "tuple semantics." We will now present a suite of cooccurrence analysis operators that can support various underlying assumptions. One way to achieve this is to group cooccurrence matches into clusters that represent a certain tuple. For example, there is a "unique-on" constraint on prof email area prof email, thus, the tuples appear twice in the matchings table, therefore receive a score of 2.

In the final stage, the Relation Constructor applies the relation multiplication will have specific knowledge about its "tuple semantics." To achieve this, we will introduce an appropriate selection of operators. We will thus motivate how the operators can support various underlying assumptions, giving us the ability to express a wide variety of applications in ER discovery. Our view is that each application is guided by a tuple function that captures the desired "tuple semantics." We will now present a suite of cooccurrence analysis operators that can support various underlying assumptions. One way to achieve this is to group cooccurrence matches into clusters that represent a certain tuple. For example, there is a "unique-on" constraint on prof email area prof email, thus, the tuples appear twice in the matchings table, therefore receive a score of 2.

In the final stage, the Relation Constructor applies the relation multiplication will have specific knowledge about its "tuple semantics." We will now present a suite of cooccurrence analysis operators that can support various underlying assumptions. One way to achieve this is to group cooccurrence matches into clusters that represent a certain tuple. For example, there is a "unique-on" constraint on prof email area prof email, thus, the tuples appear twice in the matchings table, therefore receive a score of 2.

In the final stage, the Relation Constructor applies the relation multiplication will have specific knowledge about its "tuple semantics." We will now present a suite of cooccurrence analysis operators that can support various underlying assumptions. One way to achieve this is to group cooccurrence matches into clusters that represent a certain tuple. For example, there is a "unique-on" constraint on prof email area prof email, thus, the tuples appear twice in the matchings table, therefore receive a score of 2.

In the final stage, the Relation Constructor applies the relation multiplication will have specific knowledge about its "tuple semantics." We will now present a suite of cooccurrence analysis operators that can support various underlying assumptions. One way to achieve this is to group cooccurrence matches into clusters that represent a certain tuple. For example, there is a "unique-on" constraint on prof email area prof email, thus, the tuples appear twice in the matchings table, therefore receive a score of 2.

In the final stage, the Relation Constructor applies the relation multiplication will have specific knowledge about its "tuple semantics." We will now present a suite of cooccurrence analysis operators that can support various underlying assumptions. One way to achieve this is to group cooccurrence matches into clusters that represent a certain tuple. For example, there is a "unique-on" constraint on prof email area prof email, thus, the tuples appear twice in the matchings table, therefore receive a score of 2.

In the final stage, the Relation Constructor applies the relation multiplication will have specific knowledge about its "tuple semantics." We will now present a suite of cooccurrence analysis operators that can support various underlying assumptions. One way to achieve this is to group cooccurrence matches into clusters that represent a certain tuple. For example, there is a "unique-on" constraint on prof email area prof email, thus, the tuples appear twice in the matchings table, therefore receive a score of 2.
using examples from the CSContact and CSResearch applications.

To begin with, as Section 3 motivated from our dual hypotheses, we are to realize our tuple function through pattern-based cooccurrence analysis. Thus, to support the two aspects of both pattern matching and cooccurrence-analysis scoring, each operator specifies a “search pattern” and a “scoring measure,” which are processed by the Pattern Matcher and Tuple Ranker (Figure 3b, respectively. Specifically, each operator is written as \( \alpha(X) \), where \( X \) is a list of search terms, as either abstract entities or literal keywords, and \( \alpha \) is a pattern measure specifying how the terms are connected into a pattern.

- **Pattern matching**: \( \alpha(X) \) specifies a search pattern, in which \( X \) represent a list of search terms, as either abstract entities or literal keywords, and \( \alpha \) is a pattern measure specifying how the terms are connected into a pattern.

- **Cooccurrence-analysis scoring**: \( \beta \) is a scoring measure, which determines, upon all the matched occurrences, the specific scoring method.

**Example 5 (Operator Format)**: Recall the tuple function in Figure 8: \( F = \#tf-uw50(#entity(prof) #entity(univ)). \) In this operator, \#tf specifies a scoring measure “tuple frequency,” and \#uw50 specifies a pattern measure in which prof and univ must appear within 50 words of each other. Note our query syntax uses \#entityE to specify that \( E \) represents an abstract entity (and not a literal keyword). To contrast, if we are to discover (prof univ) for a prof who received a PhD from univ, the tuple function could be \#tf-uw50(#entity(prof) phd #entity(univ)), in which we use keyword phd as part of the search pattern.

Thus, as a uniform format, each operator is comprised of two components: a search pattern \( \alpha(X) \) and a scoring measure \( \beta \). As an overview, Figure 9 and Figure 10 summarizes our currently supported pattern measures and scoring measures, respectively. Note that, in our realization of pattern-based cooccurrence analysis, an operator \#\( \beta - \alpha(\cdot) \) can be “constructed” by any combination of a scoring measure \( \beta \) with a pattern measure \( \alpha \). To introduce both aspects of the operator, we will discuss pattern matching measures in Section 5.1 and cooccurrence scoring measure in 5.2.

### 5.1 Pattern-Measure Techniques

This section will present our pattern measures in Figure 9. As a starting point, we will first motivate with the most basic type of pattern matching technique.

#### Document Cooccurrence

Specifically, the first search pattern that we considered was document cooccurrence, where search terms are constrained to occur within the same page, or “document.” However, after performing accuracy analysis on a variety of relations, we found that document cooccurrence yields unacceptable accuracy – generally below 50%.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{own} )</td>
<td>unordered window of size ( n )</td>
</tr>
<tr>
<td>( \text{own} )</td>
<td>ordered window of size ( n )</td>
</tr>
<tr>
<td>( \text{nnown} )</td>
<td>nearest neighbor unordered window of size ( n )</td>
</tr>
<tr>
<td>( \text{nnown} )</td>
<td>nearest ordered neighbor window of size ( n )</td>
</tr>
</tbody>
</table>

**Figure 9: Pattern measures.**

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{tf} )</td>
<td>tuple frequency</td>
</tr>
<tr>
<td>( \text{tfdj} )</td>
<td>tuple frequency, multiplied by the inverse document frequency of the ( j )-th entity</td>
</tr>
<tr>
<td>( \text{dist} )</td>
<td>distance weighted</td>
</tr>
</tbody>
</table>

**Figure 10: Scoring measures.**

This is a result of the fact that many unrelated terms cooccur in the same document. For instance, university phone numbers, common research area such as Algorithms, and email addresses such as webmaster@cs.uiuc.edu and colloq@cs.wisc.edu tend to occur in many documents.

In addition, on hub pages, or pages that list a number of entity instances, unrelated instances will match the document cooccurrence pattern. An example of a hub of UIUC computer science professors appears in Figure 11. From the figure, hubs tend to list related entities in localized groups, however, this locality can not be expressed with document cooccurrence. In general, we believe that terms that cooccur together with some locality are related to each other. However, document cooccurrence does not specify enough locality between entities for accurate relation discovery.

#### Window of Words

Therefore, we will consider a window of words, where the terms in the tuple function must occur within a specified number of words. There are two window of words pattern matching techniques: unordered window of words and ordered window of words, which have the syntax \( \text{uw50} \) and \( \text{own} \), respectively. For a tuple function using an unordered window of words, \#\( \beta - \text{uw50}(E_1 E_2 \ldots E_n) \), and a collection of Web documents with instances \( e_i \) of the entity \( E_i \) at position \( e_i.\text{pos} \), a pattern \( e_1, e_2, \ldots, e_n \) will match if and only if:

\[
\forall i, j | i \neq j \text{ and } e_i.\text{pos} - e_j.\text{pos} <= n
\]

For an ordered window of words query, \#\( \beta - \text{own}(E_1 E_2 \ldots E_n) \), a pattern \( e_1, e_2, \ldots, e_n \) will match if and only if:

\[
\forall i, j | i \neq j \text{ and } e_i.\text{pos} < e_j.\text{pos} \land (e_j.\text{pos} - e_i.\text{pos} <= n)
\]

Both window of words pattern matching techniques are common to many traditional search engines. Applications can choose an appropriate ordering constraint based on the relation being discovered. For instance, with the CSContact application, people generally list their name before their email address or phone number. Therefore, an ordered window operator is probably more appropriate to constrain an ordering for prof-email and prof-phone occurrences. On the other hand, an unordered window is probably more appropriate for the CSResearch application, because there is no obvious ordering between a professor’s name, research area, and university. Therefore, the user would not want to restrict the patterns found based on an arbitrary ordering. Thus, for discovering (prof, phone, email) in CSContact, it makes sense to use \#\( \text{tf-uw50(#entity(prof) #tf-uw50(#entity(phone) #entity(email)))} \). This is because there is no clear ordering between phone number and email address, however, there is a logical ordering relative to the professor’s name.

#### Nearest Neighbor Window of Words

Although a window of words can specify more locality than document cooccurrence, a basic window can not handle all applications
equally well. For instance, if the application is trying to discover an implicit 1-to-1 mapping between entity instances, a basic window of words may not perform well. The problem can arise on a hub page because the Pattern Matcher will match all combinations of entities that appear within the specified window, however, only the first match may be valid. For instance, pages such as the hub of UIUC computer science professors, shown in Figure 11, could cause too many matches for the CSContact application. Specifically, each professor instance may pattern match with several other professors’ email addresses and phone numbers.

Thus, we have developed nearest neighbor window of words techniques: \( \text{nnw} \) and \( \text{nwn} \). The nearest neighbor further constrains the window of words operators beyond Equation (1) or Equation (2). Specifically, a tuple with entities \( E_1, E_2, ..., E_n \) and instances of the entities \( e_{1,1}, e_{1,2}, ..., e_{n,1}, e_{n,2} \) will pass the nearest-neighboring constraint if and only if:

\[
\forall_{i,j,v,w} \left( [e_{i,x}.pos - e_{j,y}.pos] < [e_{i,v}.pos - e_{j,y}.pos] \right) \land \left( [e_{i,x}.pos - e_{j,y}.pos] < [e_{i,x}.pos - e_{i,j,w}.pos] \right)
\]

5.2 Scoring-Measure Techniques

We next address the issue of cooccurrence scoring—After matching pattern occurrences of entity instances \( e_1, ..., e_n \) across Web pages, how to score the corresponding tuple \( \langle e_1, ..., e_n \rangle \)? This section will motivate the tuple scoring techniques in Figure 10.

Tuple Frequency

The most basic scoring measure is a tuple frequency measure, which has the syntax \( \text{tf} \). As demonstrated in Section 4.2, the tuple frequency is calculated by counting the number of times a tuple, \( \langle e_1, ..., e_n \rangle \), appears in the matchings table. For many applications, a simple count is sufficient, however, this technique can be too simplistic for other applications. For example, a term-frequency based approach tends to give high scores to common email addresses, phone numbers, etc.

Tuple Frequency - Inverse Document Frequency

In order to handle entities that are unevenly distributed across the Web, we support a TFIDF scoring measure, which has the syntax \( \text{tfidf} j \). This measure is calculated by multiplying the tuple frequency of a tuple \( \langle e_1, ..., e_n \rangle \) by the inverse document frequency of the entity instance \( e_j \). The inverse document frequency, \( \text{IDF}(e_j) \), can be calculated from the document frequency, \( \text{DF}(e_j) \), which is the number of documents that \( e_j \) appears in at least once [17]. Then, with \( |D| \) documents, the inverse document frequency is:

\[
\text{IDF}(e_j) = \log\left( \frac{|D|}{\text{DF}(e_j)} \right)
\]

(4)

For example, assume the tuple function \#tfidf2(#entity-(prof) entity(univ)) was applied to the example in Figure 4. Then, the tuple (David DeWitt, U. of Wisconsin) would receive a score of .325. Specifically, the tuple frequency remains 2, while the inverse document frequency of the university instance, University of Wisconsin, is \( \log(\frac{2}{4}) = .176 \), as the University of Wisconsin appears in D1 and D3 of the DPI list.

The TFIDF that we propose is a variation of a common technique used in information retrieval [17]. Specifically, the TFIDF measure in information retrieval has no concept of a tuple. Rather, the TF refers to the frequency of a term within a document. Thus, TFIDF is traditionally used to measure the importance of a term within a document. However, the motivation remains the same, which is that the occurrence of a term is not as meaningful if it occurs very frequently. Thus, this scoring technique is useful for applications where entity instances should be evenly distributed across tuples, but they are not evenly distributed across the Web.

An example of this is the CSContact application, where emails and phone numbers should be evenly distributed across professors, however, that is not the case across the Web. On the other hand, with the CSResearch application, research areas are not necessarily evenly distributed across the Web, but we probably do not want to use a TFIDF scoring measure. Specifically, research areas are not evenly distributed across professors, where there are more professors with Architecture or Database Systems research interests than professors with Compilers or User Interface research interests. Therefore, depending on the application, a term-frequency measure may be preferable to a TFIDF measure.

Distance Weighted

Another scoring measure is a distance-weighted scoring measure, with the syntax \( \text{dvw} \), where the tuples are scored by a root mean square distance measure. Assume each tuple, \( t_j \), has instances of entities \( e_1, e_2, ..., e_n \), where \( e_1.pos < e_2.pos < ... < e_n.pos \), then the root mean square distance is:

\[
D(t_j) = \sqrt{\frac{\sum_{i=1}^{n-1} (e_i.pos - e_{i+1}.pos)^2}{n - 1}}
\]

(5)

Then, each tuple, \( t_j \), will receive a score of:

\[
\frac{1}{D(t_j)}
\]

(6)

This root mean square distance measure is used in text retrieval with the motivation that terms that occur closer together are more related [14]. This is a similar motivation to that of the nearest neighbor technique. The difference between the two methods is that the nearest neighbor technique is context-sensitive, omitting entity instances from consideration based on the surrounding entity instances. The nearest neighbor technique is useful for hub pages, as discussed in Section 5.1, however, this can be too restrictive for other situations.

For example, in the case of (prof research) discovery, a professor may have multiple research areas. In this case, these multiple instances of research (e.g., programming languages, formal systems, etc.) will appear next to prof, but typically in order of importance. Therefore, the distance-weighted technique may be more appropriate, as all of the research areas will be pattern matched. Also, the distance-weighted measure reduces sensitivity to the window size, as instances that are far apart will receive a lower weighting.

As we conclude, we have described both pattern-measure \( \alpha \)'s (Figure 9) and scoring-measure \( \beta \)’s (Figure 10) for together constructing our tuple function \( F=\#\beta\alpha(X) \). As the core of ER discovery for ranking tuples, these tuple functions are supported in our WISDM-ER system, by the Pattern Matcher (for pattern matching) and the Tuple Ranker (for tuple scoring), as we discussed in Section 4. Our system development is thus complete, with the realization of this suit of operators.

6. PROTOTYPE AND CASE STUDIES

Toward our goal of providing indexing and search service for Web mining applications, we have built our prototype WISDM-ER system for ER discovery. Section 6.1 will discuss our prototype tested. Further, to demonstrate its usage, Section 6.1 presents two real “case studies,” to show the possibilities and effectiveness of building Web mining upon WISDM-ER. We demonstrate two sample applications—CSContact and CSResearch—as “benchmark” scenarios we have used throughout. For further reference, we publish the system for real-time online demo at wismd-er.myftp.org."
6.1 System Prototype

We now report our implementation of our proposed system architecture (Section 4). We will discuss the implementation of the six components of our ER discovery system in Figure 3: Web Crawler, Entity Extractor, Inverted-List Indexer, Pattern Matcher, Tuple Ranker, and Relation Constructor. As our current testbed focuses on supporting interesting discovery relating to computer science in an academic setting (i.e., our benchmark scenarios CSContact and CSResearch), our testbed collects, extracts, and indexes Web pages from several computer science departments. As our general platform, we have implemented the system in C++ on Red Hat Linux with gcc 3.2.2, unless otherwise noted.

As Section 4 describes, we build our new ER discovery system upon a traditional page retrieval search engine (“almost for free”). Specifically, we have extended the Lemur Toolkit (version 2.2), an information retrieval engine [1]. We extended the system as Figure 3 shows (and as Section 4 described). On one hand, at the “bottom,” as Lemur is keyword-based and not entity-aware, we extended the Inverted-List Indexer and Pattern Matcher to support entities. On the other hand, at the “top,” as Lemur is document-based and not relation-aware, we implemented our new components of the Tuple Ranker and Relation Constructor.

To support ER discovery, as our “virtual” Web Crawler, we obtained a portion of a Web crawl from the Stanford WebBase Project*. Currently, we have indexed a crawl from January 2004 of six universities. This crawl contains 82,453 web pages and 1.4GB of raw text, as Figure 12 summarizes. We are in the process of scaling up our corpus to focused crawls of some 20 CS departments from the WebBase group—We are grateful to their generous support.

Finally, for entity extraction, we have implemented two types of Entity Extractors (or “taggers”)—Section 7 provides more details on different types of taggers and their suitability for different kinds of entities. First, our pattern-driven extractor encodes rules for tagging entities with regular patterns—e.g., phone and email. Second, our dictionary-driven extractor works for entities whose domains, or dictionaries, are enumerated (e.g., a list of professors, a list of states)—e.g., university (those in Figure 12), professor (at these universities), research (as areas in CS), and state (the US states).

We stress that, while not a focus of our work, our tagging is rather scalable—it reads all of the “dictionaries” into a hash table and scans the text in just one pass, during the same time as offline text indexing. To give a perspective, in comparison, the Entity Extractor runs in under half of the time of Lemur’s Inverted-List Indexer. In our own experience as well as several related efforts, such pattern and dictionary-driving tagging have proven effective—For instance, using the TAP ontology [10] (which, unlike our simple dictionary, is a sophisticated “knowledge base”), SemTag [8] has semantically marked-up 264 million pages and generated 434 million semantic tags with 82% accuracy, which is the largest scale effort to date. In the meantime, to support more “application-generic” entities, such as person, location, and company, we are in the process of incorporating automatic learning-based taggers.

6.2 Case Studies

Table: Web crawl of CS departments at six universities.

<table>
<thead>
<tr>
<th>university</th>
<th>num pages</th>
<th>raw size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IIT</td>
<td>1305</td>
</tr>
<tr>
<td>2</td>
<td>Illinois</td>
<td>26158</td>
</tr>
<tr>
<td>3</td>
<td>Indiana</td>
<td>6265</td>
</tr>
<tr>
<td>4</td>
<td>Michigan</td>
<td>18982</td>
</tr>
<tr>
<td>5</td>
<td>Purdue</td>
<td>10002</td>
</tr>
<tr>
<td>6</td>
<td>Wisconsin</td>
<td>19741</td>
</tr>
</tbody>
</table>

Figure 12: Web crawl of CS departments at six universities.

It is our goal to provide relation discovery on the Web for supporting a layer for Web mining applications to be built on top (as Figure 1 shows). Therefore, to demonstrate such possibilities, this section studies two “benchmark” scenarios—our example applications of CSContact and CSResearch. As we have used these scenarios throughout the paper, we believe it is most meaningful to study and evaluate the actual working of these sample applications.

Specifically, we will report and evaluate the results of a series of sample queries for each scenario. In Figure 13, we describe the purpose of each query, the tuple function, and the three relation constraints—We have executed these queries through the system interface (Figure 5, as Section 2.2 explained). (Thus, these queries can be submitted to our online demo for their actual results.)

Note that our case studies focus on the “semantic” issues and do not discuss the “time” performance. Since our system directly builds upon an IR system (which can be any scalable search engine), such performance evaluation will likely boil down to the choice of the underlying engine—which is a Lemur. While not explicitly measured, in our experiments, the response time has been rather satisfactory even for interactive querying. We invite the readers to view the online real-time demo. The current server is running on a Pentium-4 2.6GHz PC with 1GB memory.

Application: CSContact

First, we report the CSContact application. To begin with, suppose we are interested in the phone number for David DeWitt. From the query C1a, WISDM-ER produces the result as shown in Figure 14, which does match David DeWitt’s phone number.

To contrast, now, suppose we want the “fax” number instead. Although the same schema (prof, phone) as C1a, this query has different underlying tuple semantics (we are looking for a different phone). However, while fax and phone are not distinguished in entity extraction, we can do so by WISDM-ER online disambiguation with an appropriate tuple function. Specifically, C1b accomplishes this disambiguation by adding keyword fax to the tuple function, which leads to the discovery of a different tuple, as Figure 14 shows. Contrasting C1a with C1b, we observe that such online disambiguation is guided by different tuple functions for discovering different tuple semantics—which is in essence consistent with our view of the hypotheses in Section 3.

Such online disambiguation for the desired association of instances, can be achieved in many ways—e.g., with the presence of “context” entities: Consider finding email for AnHai Doan. First, C2a produces, as Figure 14 shows, all his emails from three universities that he has been associated with. To disambiguate, however, if we know the additional context that he is at Illinois, we can perform online disambiguation to focus on only this context. Query C2b thus specifies a tuple function by adding university, which is restricted by a reference-only constraint to Illinois—and it indeed produces the right email in Figure 14.

To more systematically measure the “accuracy” this application, we evaluated a “larger” query returning many tuples (unlike previous examples). We created query C3 for finding (professor,
These results are particularly impressive because the associations obtained are not false, and they are not displayed in the database. To find the professor's name, we used a database query. The results are still quite high, showing the promise of this approach.

As another scenario, we are interested in comparing research areas. To find the research areas of a pair of universities, we developed a comparison application for the research areas of all the professors at different universities. Overall, we obtained an overall accuracy of .88, as Figure 17 summarizes in the detail.

To move on, as more complex relations, we are interested in all professors with "database"-related research areas, across universities. Thus, query \( Q_3 \) produces the result, by restricting the search results. We then study the CSResearch application. As simple cases, we developed a comparison application for the research areas of all the professors at different universities, such Wisconsin and Illinois. From query \( Q_2 \), we obtain a result of very high recall and precision, which returned correct results, as Figure 14 shows.

Finally, we analyze the accuracy of the Tuple Discovery application. As another scenario, we are interested in finding the reference of an expert, for finding the relation that will plot the results in a bar chart, as Figure 17 also shows.

Next, we will study the CSResearch application. As simple cases, we developed a comparison application for the research areas of all the professors at different universities, such Wisconsin and Illinois. From query \( Q_2 \), we obtain a result of very high recall and precision, which returned correct results, as Figure 14 shows.

Overall, we obtained an overall accuracy of .88, as Figure 17 summarizes in the detail. As the "truth," we manually collected the correct tuples for all CS professors at the three universities. These results are particularly impressive, because the associations obtained are not false, and they are not displayed in the database. To find the professor's name, we used a database query. The results are still quite high, showing the promise of this approach.
between all three entities must be correct. We also have reason to believe with a larger corpus, we will be able to filter out even more noise to produce better results.

7. RELATED WORK

The goal of WISDM-ER is to provide a generic, systematic Web indexing and search mechanism to support entity-relation discovery and then to benefit the development of many Web-mining applications. To our knowledge, there is little work directly addressing the problem we consider in this paper. In this section, we review some works that are considered relevant to ours in the following aspects: semantic entity and relation annotation, and cooccurrence-and pattern-based relation mining.

Sharing a similar goal of building a more structured and machine-understandable Web, many research studies are addressing the problem of semantic annotation on Web pages, e.g., Semantic Web. Most of such studies perform the tagging of relations through the use of carefully crafted ontologies [8, 11, 18]. For example, SemTag [8] tags entities and relations using the TAP ontology [10]. However, the creation and maintaining of ontologies is arduous and time-consuming. In contrast, our approach uses holistic cooccurrence analysis to dynamically and reliably discover the association between entities. In addition, as the entities (such as universities) are rarely changed but the relations (or tuple semantics) to discover vary enormously and depend on individual application, relying on those “static” relations defined in the ontology is rather inflexible. Our approach aims for an adaptable search-based support towards mining relations, beyond just annotating relations.

One core technique in our tuple discovery is cooccurrence analysis. As the Web becomes the largest and most ubiquitously available data repository in the world, cooccurrence analysis based on such Web-scale statistics have been explored in many problems, such as finding synonyms in an IR system [19], validating the question-answer pairs in a Q/A system [14, 15], and acquiring hit counts to support various mining tasks in databases [9, 12]. However, most of such works exploit cooccurrence analysis in a rather ad-hoc way: First, the lack of abstraction makes their approaches rather task-specific and hard to adapt; Second, they only provide limited operations, such as the use of hit counts of keywords or documents—whatever the search engines provides; And third, due to the previous two points, they lack systematic search-based support. In contrast, our work aims to provide a systematic mechanism to support various cooccurrence scoring techniques on top of the various cooccurrence patterns. With such systematic search support on pattern-based cooccurrence analysis, many applications can be easily built, including those just mentioned.

Entity and relation extraction is a traditional problem in information extraction (IE). Entity extraction, as one building block of our system, has been explored extensively and is rather mature. The major problem of applying such approaches, as shown in [5], is that the patterns learned in later steps become enormous and error-prone so that the tuples extracted become less and less reliable. In contrast, we use the cooccurrence analysis to find the matching relation tuples. The large scale of Web data becomes an opportunity, instead of obstacle, to our approach. In addition, our system is a combination of entity extraction and relation mining, while earlier works generally address only one of the tasks.

8. CONCLUSION

We have introduced ER discovery as a useful function that supports mining of the Web by providing a layer of knowledge representation. In addition, we have implemented WISDM-ER by extending a traditional IR search engine, which shows interesting applications of ER discovery and 83% – 91% accuracy. We will continue to build upon our current system to support a broader spectrum of Web mining applications. First, we plan to integrate machine learning based entity taggers, in order to support generic entities such as person and organization. In addition, we will consider stop-word conditions, to retrieve promising relations on a much larger scale. Finally, we will consider more advanced Web mining applications on top of WISDM-ER.

9. REFERENCES