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 sifting 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 testbed, 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 *knowledge discovery*– On the one hand, tackling the challenge, search engines (*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 sifting 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 "preprocessing" 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.

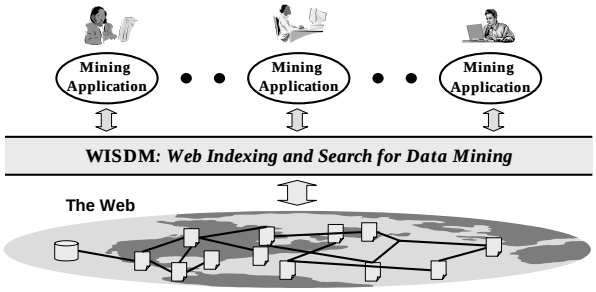


Figure 1: *WISDM*: Our ultimate goal.

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*.

.

email = dewitt@cs.wisc.edu.

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 $R1$ and $R2$, 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 prof, phone, email into a relation ⟨prof, phone, 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 prof, univ, research into a relation ⟨prof, univ, 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.*:

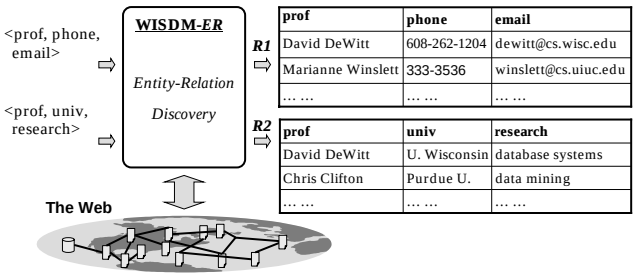
- By joining $R1$ with $R2$: What are the emails of the database professors at Wisconsin?
- By joining $R2$ with a "university ranking" database: Which top-20 university has the most database faculty?

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.

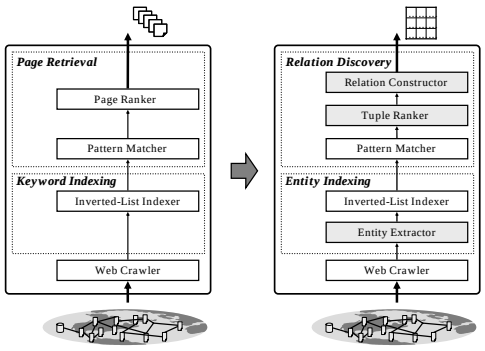
Toward building *WISDM-ER*, we have developed a functioning prototype, upon the Lemur text engine [1]. With CSContact and CSResearch as our driving "benchmark" applications, our testbed



prof	phone	email
David DeWitt	608-262-1204	dewitt@cs.wisc.edu
Marianne Winslett	333-3536	winslett@cs.uiuc.edu
… …	… …	… …

prof	univ	research
David DeWitt	U. Wisconsin	database systems
Chris Clifton	Purdue U.	data mining
… …	… …	… …

Figure 2: Entity-relation discovery.



(a) Traditional page retrieval system (b) Our *ER* discovery system

Figure 3: From page retrieval to relation mining.

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 testbed** 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

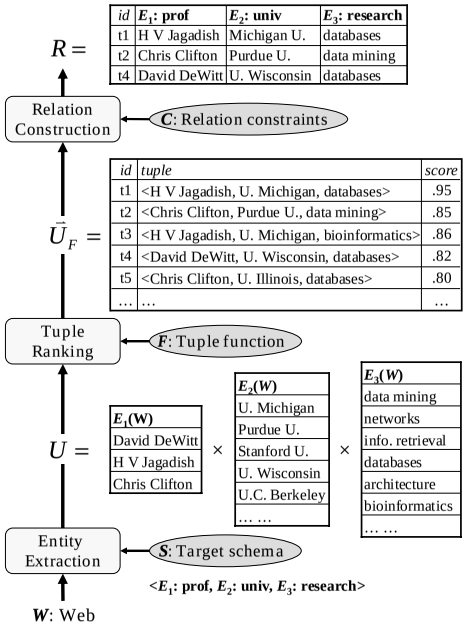


Figure 4: *ER* discovery: Overview.

associating "entities" appearing on W into "tuples," as guided by certain "objective parameters" for capturing the application semantics. Our discover is thus, given W, to weave its scattered entities into coherent relations. Note that, like any mining tasks, we should interpret a discovered tuple as a "promising" (*i.e.*, likely correct) association of entity instances. The requesting application may perform further specific processing to complete its mining task.

2.1 Overview

To generally abstract this task, what does such discovery consist of? Essentially, *ER* discovery is guided by three objective parameters: 1) *target schema* S, which defines what entities to look for, 2) *tuple function* F, which determines if certain instances of the specified entities can bond to form a desirable tuple, and 3) *relation constraints* C, which restricts certain properties of the target relation. Guided by the three objective parameters, conceptually, *ER* discovery consists of three progressive steps: extracting entities, ranking tuples, and finally constructing relations. As Figure 4 illustrates, we will walk though the construction of ⟨prof, univ, research⟩ (*i.e.*, our CSResearch scenario; Section 1) as our running example.

2.1.1 Entity Extraction

Guided by a target schema S, *ER* discovery starts with identifying a search space of possible tuples, from which the target relation can be constructed. The schema specifies what entities, or "attributes," will constitute the target relation, *i.e.*, $S = \langle E_1, \ldots, E_n \rangle$, where each E_i is an entity. For our example (Figure 4), $S = \langle E_1$:prof, E_2:univ, E_3:research$\rangle$.

As the first conceptual step, *entity extraction* is to identify what literal instances each entity can represent. Like an attribute in a relational table, an entity represents literal values of the same semantic type; *e.g.*, entity univ can represent, say, `U. Wisconsin`, `Stanford U.`, and others. Over the Web W, an abstract entity E_i can appear as various concrete *instances*, which together form the domain E_i(W) of the entity. (Note that, in practice, E_i(W) is not necessarily "exhaustive"– it only needs to cover instances the application is interested in.)

The goal of entity extraction is thus to ensure that every E_i referenced in S is well defined– to construct E_i(W) by extracting its instances from the text on the Web. Such extraction finds not only "what" an instance is but also "where" it is mentioned on the Web (*e.g.*, instance `David DeWitt` may occur in different pages). In principle, such extraction can happen *online*, after S is given at runtime, or *offline*, before any queries. However, in practice, for realizing *ER* discovery as an efficient index-based search mechanism, entity extraction will likely be "compile time." In our implementation, as Section 4 will discuss, this step indeed happens at the time of indexing. (Section 7 will discusses more extensively how we can build *ER* discovery upon entity-tagging techniques.)

With each E_i(W) extracted, the schema S "spans" a space of candidate tuples, as the "search space" for constructing the target relation. More precisely, with $S = \langle E_1, \ldots, E_n \rangle$, *ER* discovery is to select tuples from $U = E_1(W) \times \cdots \times E_n(W)$, which we call the *tuple universe*.

Example 1 (Entity Extraction): Consider the CSResearch scenario (Figure 4), with $S = \langle E_1$:prof, E_2:univ, E_3:research$\rangle$, our universe $U = E_1(W) \times E_2(W) \times E_3(W)$. It consists of every combination of prof, univ, and research, *e.g.*, ⟨`David DeWitt`, `Illinois`, `bioinformatics`⟩. Our challenge of *ER* discovery is thus to search for promising tuples in this large space. ■

We stress that the target schema S, which our *ER* discovery starts with, captures our goal of knowledge discovery. With an objective schema of $S = \langle E_1, \ldots, E_n \rangle$, we are effectively asking– How can the instances of entities $E_1, \ldots, E_n$ associate with each other? As mentioned earlier, such associations generally translate to various information need– Our core task of *ER* discovery is to find promising associations, which will help fulfilling the need.

2.1.2 Tuple Ranking

Given the tuple universe as S spans, what are the tuples that are "promising" for the target relation? Clearly, while arbitrary combinations of instances from E_i(W) are possible, not all of them are meaningful– We note that such meaning or *tuple semantics*– that whether certain association of entity instances form a tuple– depends on specific applications. For instance, for our CSContact application, for ⟨prof, phone, email⟩, what should be the phone associated with a prof? (*e.g.*, Office phone? Home? Fax number?) How about email? (*e.g.*, there may similarly be multiple emails per prof). The core challenge of *ER* discovery clearly lies in effectively finding such tuples, as desired by the applications.

By the second conceptual step, *tuple ranking*, we view the discovery of desired tuples as ranking the tuple universe U, resulting in a ranked list $\vec{U}_F$, guided by some *tuple function* F as an objective parameter that the application specifies. (We use $\vec{U}_F$ to denote an ordered "vector" version of U, with respect to F.)

The tuple function F induces a "score" for each tuple and, accordingly, a ranking of universe U. To quantify the "promise" of a tuple, we view F as a function that, given tuple $t \in U$, returns a *tuple score* $F(t)$. Thus, for U as a universe with $S = \langle E_1, \ldots, E_n \rangle$:

$$F : E_1(W) \times \cdots \times E_n(W) \longrightarrow \mathbf{R}$$

While such a ranking depends on applications, *ER* discovery should support the construction of an effective tuple function and its efficient evaluation. As a core challenge of this paper, to realize *ER* discovery, we must derive adequate abstraction and search techniques for tuple ranking (for which we will develop in Section 3 and 4 respectively).

Example 2 (Tuple Ranking): Continuing our CSResearch scenario, given U (Example 1), the tuple ranking step ranks these tuples with a tuple function F– Let's assume that the application has specified

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 $\vec{U}_F$ as $(t_1, t_2, t_3, \ldots)$, with scores $F(t_1)$ = .95, $F(t_1)$ = .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 $t1$ and $t3$ on `H V 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 → 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 → univ (although prof ↛ 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 $\vec{U}_F$ (as the result of tuple ranking), select the first tuple of every prof, and construct $R = \{t_1, t_2, t_4\}$. Thus, $t3$ is not included, as it "duplicates" t_1 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, ..., 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 univ ⊆ {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 = `#dist-uw100(...)`. (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 = ⟨professor, university, phone⟩.

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={professor is a key; university ⊆ {U. Wisconsin}}.

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*

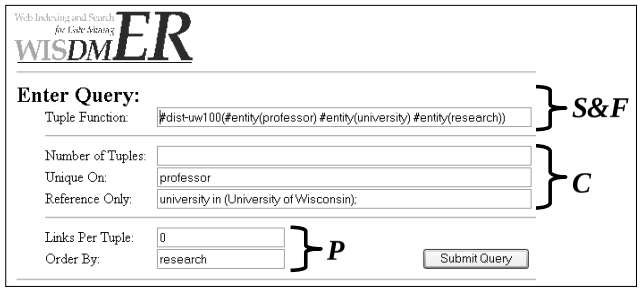


Figure 5: Our *ER* discovery system query interface.

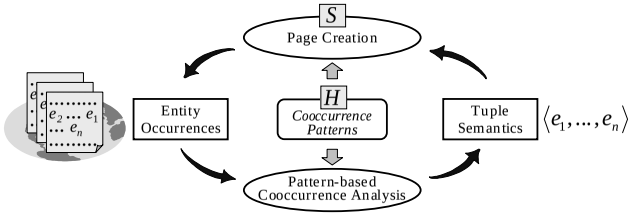


Figure 6: Hypotheses: Discovering tuple semantics.

Our task of tuple ranking, as a core step in *ER* discovery, is to discover promising tuples from the potential associations of entity instances. As Figure 4 shows, for a target schema $\langle E_1, \ldots, E_n \rangle$, the tuple universe or the "search space" contains all possible combinations of entity instances– *i.e.*, the cartesian product of E_i(W). What $e_1, \ldots, e_n$, as instances for each E_i, *i.e.*, $e_i \in E_i(W)$, can actually form a "meaningful" tuple $\langle e_1, \ldots, e_n \rangle$? Such "meaningfulness" of entity associations– which we call *tuple semantics*– is not only implicit but also depends on specific applications. As a critical challenge to enabling *ER* discovery, is the discovery of tuple semantics possible?

As an essential foundation of *WISDM-ER*, and a main thesis of this paper, we propose to realize tuple ranking by *pattern-based cooccurrence analysis* across the large scale of Web pages. We believe that, for our challenge to "weave" entities into tuples for constructing relations, such associated entities often materialize themselves as cooccurring terms in Web text. This section develops this insight.

To begin with, while our discovery attempts to find meaningful entity associations that are "deeply" underlying Web pages, as our input, we can only "shallowly" observe how entities occur, by way of text presentation, on the Web. That is, we are to discover underlying tuples (*e.g.*, ⟨`David DeWitt`, `U. Wisconsin`⟩) behind surface clues of how entities (*i.e.*, `David DeWitt`, `U. Wisconsin`) appear on Web pages. We observe that, as we will discuss, there seems to be some "connections" between such underlying tuples (which we are to discover) and their Web presentations (which we can observe). To capture this insight, we thus make two hypotheses, which relate underlying *tuple semantics* to observable *entity occurrences*, across Web pages, as Figure 6 shows.

- **($\mathcal{S}$)** *Shallow observable clues:* An "underlying" tuple $\langle e_1, \ldots, e_n \rangle$ with certain *tuple semantics* of a Web page often connects to its "observable" presentation in some way– in terms of the *entity occurrences* in the page– This "connection" happens during *page creation* of each page.
- **($\mathcal{H}$)** *Holistic hidden regularity:* The observable occurrences of e_1, $\ldots$, e_n, at various Web pages, often share implicit *cooccurrence patterns*, which will reveal holistically when observed across many pages.

As their implications, the dual hypotheses shed light for tuple discovery over the Web: To explore certain tuple semantics, by identifying its likely cooccurrence patterns, our task is naturally the *inverse*, as Figure 6 of this semantics-to-presentations connection: Our *ER* discovery will holistically analyze the shallow clues of entity occurrences across Web pages, as guided by the cooccurrence patterns, to discover the desired tuple semantics. We thus propose *pattern-based cooccurrence analysis– i.e.*, holistic examination of Web pages to associate entity instances cooccurring in certain ways– for realizing tuple discovery.

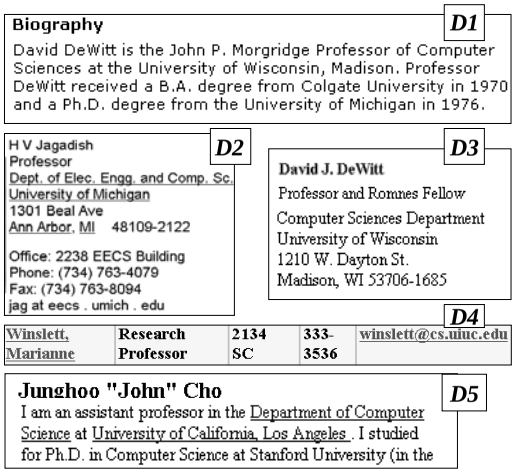


Figure 7: Example Web page snippets.

As the basis of our approach, *are these hypotheses valid?* First, for Hypothesis $\mathcal{S}$, are there certain patterns that reflect some desired tuple semantics? If such patterns emerge, the hypothesis will stand to guide our tuple discovery, by *pattern matching* for locating the "presentation" of a potential tuple. Further, for Hypothesis $\mathcal{H}$, if such patterns exist, can we observe them across a large scale? If such pattern are indeed "holistic," the hypothesis can guide us to exercise the large scale of the Web, to perform *cooccurrence analysis* across the "voting" of potential tuples.

Pattern matching: First, we observe that such patterns often naturally exist, as *conventions* to present certain knowledge. For our CSContact scenario of ⟨prof, phone, email⟩– Suppose our "tuple semantics" means associating a prof with its phone and email. We will likely see, for tuple $\langle e_1, e_2, e_3 \rangle$, that e_1, e_2, and e_3 cooccur in Web pages, near each other, with e_1 preceding the others– *i.e.*, of patterns "... prof ... phone ... email ..." or "... prof ... email ... phone ..." As examples, Figure 7 shows several Web page snippets. In particular, we observe these patterns in $D2$ and $D4$. As another example, for ⟨prof, univ⟩ (as part of CSResearch). We expect, for tuple $\langle e_1, e_2 \rangle$, that e_1 cooccur in Web pages, near each other, and in that order– *i.e.*, of pattern "... prof ... univ"– which appears in $D1$, $D2$, $D3$, $D5$.

As further evidences, we stress that several earlier works have explored such patterns, for finding certain information on the Web. In particular, several recent Web "question answering" techniques (*e.g.*, [14, 15]) work by identifying, from a target question (*e.g.*, "Where is the White House?"), a pattern that the answer may be presented (*e.g.*, "The White House is located at ..."), and use a search engine to execute pattern matching for locating "answering pages" (which will mention, say, "The White House is located at 1600 Pennsylvania Avenue in Washington, DC").

Cooccurrence Analysis: Second, we believe such patterns can often be consistently observed across many pages. As the nature of the Web dictates, presentation "conventions" (as just discussed above) will not only emerge (as just discussed) but also likely to

converge at a large scale. To begin with, since the Web has become our "ultimate information source," the need of *Web usability*[2] 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 for 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 $D1$, $D2$, and $D3$. 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 stores 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

[2] *e.g.*, for sample discussion: `www.useit.com/alertbox/20040913.html`.

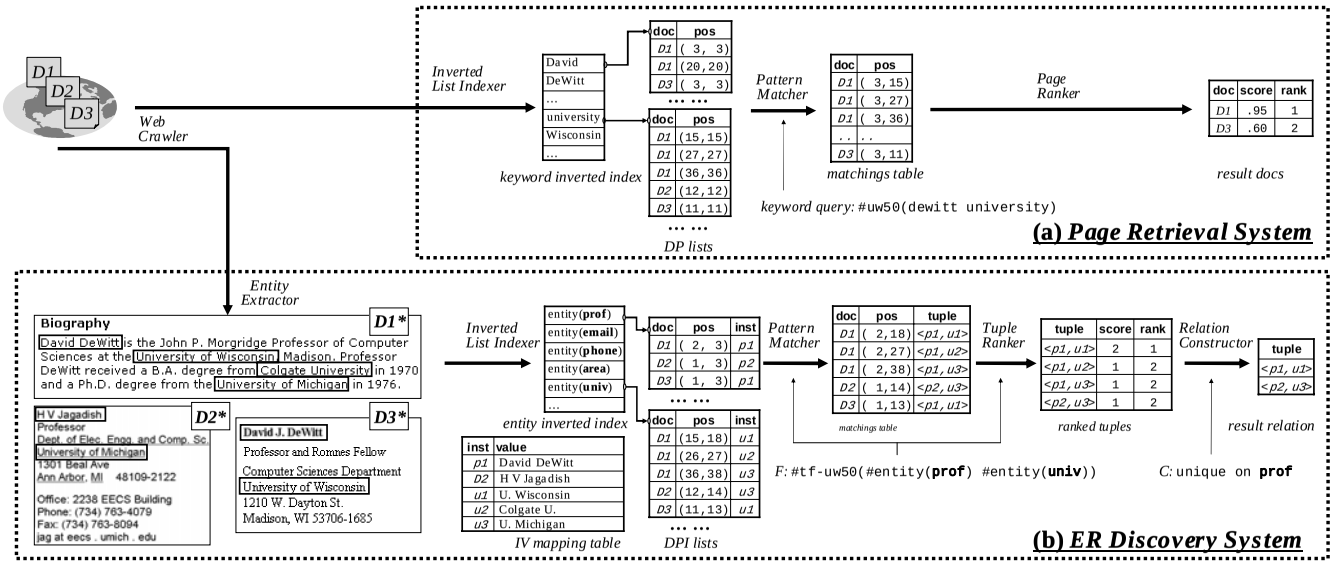


Figure 8: Query processing: From page retrieval to *ER* discovery.

calculate a score for the documents. The end result of the query is the high-ranked documents that contain the search pattern, which are $D1$ and $D2$ from Figure 4.

4.2 Extension: ER Discovery Almost for Free

Now that we have described the basis architecture, we will demonstrate how we can extend this architecture to perform ER discovery, almost for free. In order to do this, we will step through an example that is analogous to the example in Section 4.1.

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 keywords. 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 entity extraction, which is described in Section 7. From Figure 4, prof and univ entities are identified within documents $D1$, $D2$, and $D3$, resulting in augmented documents $D1$*, $D2$*, and $D2$*.

The next step to searching on abstract entities is building an *entity inverted index*. Specifically, from the augmented documents D_i^*, the Inverted-List Indexer builds an inverted index of entities, in addition a keyword inverted index as described in Section 4.1. Thus, given an entity, E_i, the entity inverted index will return a *document-position-instance list*, or DPI list. The DPI list stores the Web document IDs, word positions and entity instances for every appearance of an instance of E_i. In addition, the entity instances map to specific values in the *instance-value mapping table*, or IV mapping table, as shown in Figure 4. For example, a lookup on the inverted index for the prof entity returns a DPI list that contains documents, positions, and instances, where the instances map to `David DeWitt` and `H V Jagadish`.

Conceptually, the entity inverted index is the same as the keyword inverted index. The only difference is that since the entity inverted index is indexing abstract entities, the concrete instance values must also be stored.

Now that the crawling and indexing are completed offline, we will consider the online tuple function `#tf-uw50(#entity(`prof`) #entity(`univ`))`. As with the Inverted-List Indexer, the Pattern Matcher must be generalized to handle abstract entities. From Figure 4, this requires the matchings table to maintain the concrete instance values, which form tuples, such as $\langle p1, u1\rangle$ or $\langle$`David DeWitt, U. of Wisconsin`$\rangle$. At the core, however, the Pattern Matcher in our ER discovery system is the same as in a page retrieval system. Specifically, in both systems, the Pattern Matcher applies cooccurrence constraints by utilizing the inverted index.

Since the end goal of the ER discovery system is to discover relations, rather than basic page retrieval, the final stages deviate from a page retrieval system. Specifically, the Tuple Ranker applies a scoring function to the tuples in the matchings table. The scoring function in this example is *tuple frequency*, which is specified by `#tf-uw50` in the tuple function. The tuple frequency is calculated by counting number of occurrences of each tuple. For instance, <`David DeWitt, University of Wisconsin`> appears twice in the matchings table, therefore receives a score of 2.

In the final stage, the Relation Constructor applies the relation constraints C and returns the tuples with highest scores. In our example, there is a "unique-on" constraint on prof, thus, the tuples with the highest score for each prof are returned as the discovered relations, which are $\langle$`David DeWitt, U. of Wisconsin`$\rangle$ and $\langle$`H V Jagadish, U. of Michigan`$\rangle$.

Thus, we have described the two key extensions to support ER discovery: searching on abstract entities and constructing discovered relations. These extensions do not change the core functionality of a page retrieval system, which is performing pattern matching by applying cooccurrence constraints over the Web. Therefore, by keeping the core functionality the same, we are able to extend a page retrieval system into an ER discovery system, almost for free.

5. QUERY PRIMITIVES

As a main challenge for realizing *ER* discovery, we must achieve effective and efficient discovery of promising tuples– by ranking tuples in the universe (Section 2). As Section 3 motivated, we abstract such tuple mining as pattern-based cooccurrence analysis, as guided by a tuple function that captures the desired "tuple semantics." We will now present a suite of cooccurrence analysis operators that are used to implement this tuple function.

We aim at proposing these operators as generic for supporting a variety of applications in *ER* discovery. Our view is that each application will have specific knowledge about its "tuple semantics" leading to an appropriate selection of operators. We will thus motivate how the operators can support various underlying assumptions,

α	*description*
uw*n*	unordered window of size n
ow*n*	ordered window of size n
nnuw*n*	nearest neighbor unordered window of size n
nnow*n*	nearest neighbor ordered window of size n

Figure 9: Pattern measures.

β	*description*
tf	tuple frequency
tfidf*j*	tuple frequency, multiplied by the inverse document frequency of the j-*th* entity
dist	distance weighted

Figure 10: Scoring measures.

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 $\#\beta\text{-}\alpha(X)$, where # is an operator prefix, and executes the two functions as follows:

- **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 α is a *pattern measure* specifying how the terms are connected into a pattern.
- **Cooccurrence-analysis scoring:** β 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 #entity$E$ 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 β. 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 β with a pattern measure α. 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%.

Name	Title	Office	Phone	Email
Adve, Sarita	**Associate Professor**	**4110 SC**	**333-8461**	**sadve@cs.uiuc.edu**
Adve, Vikram	**Assistant Professor**	**4235 SC**	**244-2016**	**vadve@cs.uiuc.edu**
Agha, Gul	**Professor**	**2104 SC**	**244-3087**	**agha@cs.uiuc.edu**
Ahuja, Narendra	Affiliate Professor	2041 BI	333-1837	n-ahuja@uiuc.edu

Figure 11: Hub page for UIUC CS professors.

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 `uwn` and `own`, respectively. For a tuple function using an unordered window of words, $\#\beta\text{-}\texttt{uwn}(E_1E_2...E_n)$, and a collection of Web documents with instances e_i of the entity E_i at position e_i.pos, a pattern $e_1, e_2, ..., e_n$ will match if and only if:

$$\forall_{i,j i!=j} |e_i.pos - e_j.pos| <= n \quad (1)$$

For an ordered window of words query, $\#\beta\text{-}\texttt{own}(E_1E_2...E_n)$, a pattern $e_1, e_2, ..., e_n$ will match if and only if:

$$\forall_{i,j i!=j} (e_i.pos < e_j.pos) \wedge (e_j.pos - e_i.pos <= n) \quad (2)$$

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 `#tf-ow50(#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: `nnuwn` and `nnown`. 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_{1,m_1}, ..., $e_{n,1}$, $e_{n,2}$, ... e_{n,m_n} will pass the nearest-neighbor constraint if and only if:

$$\forall_{i,j,v,w} \quad (|e_{i,x}.pos - e_{j,y}.pos| < |e_{i,v}.pos - e_{j,y}.pos|) \wedge (|e_{i,x}.pos - e_{j,y}.pos| < |e_{i,x}.pos - e_{j,w}.pos|) \quad (3)$$

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 `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 `tfidfj`. 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, $IDF(e_j)$, can be calculated from the *document frequency*, $DF(e_j)$, which is the number of documents that e_1 appears in at least once [17]. Then, with $|D|$ documents, the inverse document frequency is:

$$IDF(e_j) = \log(\frac{|D|}{DF(e_j)}) \quad (4)$$

For example, assume the tuple function `#tfidf2(#entity-(prof) #entity(univ))` was applied to the example in Figure 4. Then, the tuple $\langle$`David DeWitt, U. of Wisconsin`$\rangle$ would receive a score of .352. Specifically, the tuple frequency remains 2, while the inverse document frequency of the university instance, University of Wisconsin, is $\log(\frac{3}{2}) = .176$, 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 `dw`, 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:

$$D(t_j) = \frac{\sqrt{\sum_{i=1}^{n-1}(e_i.pos - e_{i+1}.pos)^2}}{n-1} \quad (5)$$

Then, each tuple, t_j, will receive a score of:

$$\frac{1}{D(t_j)} \quad (6)$$

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 $\langle$prof research$\rangle$ 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 α's (Figure 9) and scoring-measure β's (Figure 10) for together constructing our tuple function F=#β-$\alpha(X)$. 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 testbed. 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– CSContactand CSResearch– as "benchmark" scenarios we have used throughout. For further reference, we publish the system for real-time online demo at `wisdm-er.myftp.org`[3].

[3]To honor the anonymity reviewing policy, instead of using our

	university	num pages	raw size (MB)
1	IIT	1305	10
2	Illinois	26158	927
3	Indiana	6265	56
4	Michigan	18982	172
5	Purdue	10002	121
6	Wisconsin	19741	117

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

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 support 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[4]. Currently, we have indexed a crawl from January 2004 of six universities. This crawl contains 82453 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 scan 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

own Web site, we register the demo at this anonymous address.

[4]http://www-diglib.stanford.edu/~testbed/doc2/WebBase/

Figure 14: Actual query results.

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 we use 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 produce 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,

	Query Description **To find:**	*Tuple Function*	*Num. Tuples*	*Unique On*	*Reference Only*
C1a	phone number of David DeWitt	#tf-ow50(#entity(prof) #entity(phone))	1		professor in (David DeWitt)
C1b	fax number of David DeWitt	#tf-ow50(#entity(prof) fax #entity(phone))	1		professor in (David DeWitt)
C2a	email of AnHai Doan	#tf-nnow50(#entity(prof) #entity(email))	3		professor in (AnHai Doan)
C2b	email of AnHai Doan at UIUC	#tf-nnow50(#entity(prof) #entity(email) #entity(univ))	1		professor in (AnHai Doan); university in (U. of Illinois)
C3	email of profs at univ. $u	#tfidf2-ow50(#entity(prof) #entity(email) #entity(univ))			university in ($u)
R1a	university of David DeWitt	#tf-nnuw50(#entity(prof) #entity(univ))		prof	professor in (David DeWitt)
R1b	research area of David DeWitt	#tf-nnuw50(#entity(prof) #entity(research))		prof	professor in (David DeWitt)
R2	database profs across univ.	#dist-uw100(#entity(prof) #entity(univ) #entity(research))		prof	research in (DB Systems, Data Mining, IR)
R3	research areas of a pair univ.	#dist-uw100(#entity(prof) #entity(univ) #entity(research))		prof	university in (U. Illinois, U. Wisconsin)
R4	research areas of prof. in univ. $u	#dist-uw100(#entity(prof) #entity(univ) #entity(research))		prof	university in ($u)

Figure 13: Case studies: Example queries tested. (*Abbreviations should be expanded in execution, e.g., prof as professor.)*

	Query $C3$		Query $R4$	
	P	R	P	R
Illinois	$\frac{54}{59}$ = .92	$\frac{54}{54}$ = 1.0	$\frac{45}{53}$ = .85	$\frac{45}{54}$ = .83
Indiana	$\frac{20}{25}$ = .80	$\frac{20}{30}$ = .67	$\frac{25}{27}$ = .93	$\frac{25}{30}$ = .83
Wisconsin	$\frac{36}{41}$ = .88	$\frac{36}{37}$ = .98	$\frac{31}{38}$ = .82	$\frac{31}{37}$ = .84
Overall	$\frac{110}{125}$ = **.88**	$\frac{110}{121}$ = **.91**	$\frac{101}{118}$ = **.86**	$\frac{101}{121}$ = **.83**

Figure 15: Accuracy: Precision and recall for $C3$ and $R4$.

Walid Aref	Purdue University	Database Systems
Bharat Bhargava	Purdue University	Database Systems
Chris Clifton	Purdue University	Data Mining
Sunil Prabhakar	Purdue University	Database Systems
ChengXiang Zhai	University of Illinois at Urbana-Champaign	Information Retrieval
Kevin C. Chang	University of Illinois at Urbana-Champaign	Database Systems
Jiawei Han	University of Illinois at Urbana-Champaign	Data Mining
AnHai Doan	University of Illinois at Urbana-Champaign	Database Systems
Geneva Belford	University of Illinois at Urbana-Champaign	Database Systems
Marianne Winslett	University of Illinois at Urbana-Champaign	Data Mining
Olvi L. Mangasarian	University of Wisconsin	Data Mining
Jeffrey F. Naughton	University of Wisconsin	Database Systems
David J. DeWitt	University of Wisconsin	Database Systems
Raghu Ramakrishnan	University of Wisconsin	Database Systems

Figure 16: Query $R2$: Professors in DB-related areas (partial).

email, university⟩ for three universities: Indiana University, Illinois, and Wisconsin. Let D be our discovered relation (as a *set* of discovered tuples) and T be the "truth" relation (as a set of correct tuples). As standard metrics, we use *precision* $P = \frac{|D \cap T|}{|D|}$ and *recall* $R = \frac{|D \cap T|}{|T|}$. As the "truth," We manually collected the correct tuples for all CS professors at the three universities.

As Figure 15 reports, we obtained an overall P=.88 and R=.91 – which is rather satisfactory as a result of automatic mining. Specifically, Illinois showed very high recall and precision, while Indiana and Wisconsin are somewhat lower. This difference can be explained, in part, due to the fact that email addresses are often disguised (for "anti-robot" extraction) on Indiana and Wisconsin pages, by not showing an @ sign or domain names. In spite of this, the accuracy numbers are still quite high, showing the promise of *ER* discovery at a large scale.

Jiawei Han	University of Illinois at Urbana-Champaign	Data Mining
Olvi L. Mangasarian	University of Wisconsin	Data Mining
James E. Smith	University of Wisconsin	Data Structures
David J. DeWitt	University of Wisconsin	Database Systems
Jeffrey F. Naughton	University of Wisconsin	Database Systems
Geneva Belford	University of Illinois at Urbana-Champaign	Database Systems
Kevin C. Chang	University of Illinois at Urbana-Champaign	Database Systems
Raghu Ramakrishnan	University of Wisconsin	Database Systems
AnHai Doan	University of Illinois at Urbana-Champaign	Database Systems
Miron Livny	University of Wisconsin	Distributed Systems
Indranil Gupta	University of Illinois at Urbana-Champaign	Distributed Systems
Barton P. Miller	University of Wisconsin	Distributed Systems

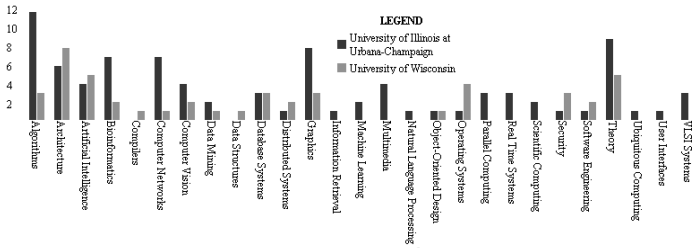


Figure 17: Query $R3$: Comparison of Wisconsin and Illinois.

Application: CSResearch

Next, we will study the CSResearch application. As simple cases, we found DeWitt's university and research with query $R1a$ and $R2a$, which returned correct results, as Figure 14 shows.

To move on, as more complex relations, we are interested in all professors with "database"-related research areas, across universities. Thus, query $R2$ produces the result, by restricting research to reference-only {Database Systems, Data Mining, Information Retrieval}. As Figure 16 shows a snippet, the results are good.

As another scenario, we are interested in comparing research areas of all the professors at different universities, such Wisconsin and Illinois. From query $R3$, we obtain a result relation covering both universities– Figure 17 (top) shows part of the result that deals with databases. To build a "comparison" application for the area distributions between universities , we developed a visualization that will plot the results in a bar chart, as Figure 17 also shows. Note that, with the support of *ER* discovery by *WISDM-ER*, such online "mining" applications can be easily realized.

Finally, we analyze the accuracy of $R4$, for finding ⟨prof, univ, research⟩. As the "truth" to compare to, we note that each prof has a unique univ but often has multiple research areas– thus we compare to their top three areas (manually compiled). Overall, we obtained P = .86 and R = .83, as Figure 15 summaries in the detail. These results are particularly impressive, because the associations

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 techniques are mainly of three types, and each helps for a different type of entity in our system: (1) pattern/rule-based– suitable for entities with generic patterns, such as email and phone; (2) dictionary-based [7]– suitable for application-specific entities, such as prof and movie; And (3) machine learning-based (e.g., Hidden Markov Model [4])– suitable for generic entities with complex patterns, such as person and address.

In addition to entity extraction, relation extraction is also considered in IE. However, most of the approaches require hand-crafted rules or training data. Wrappers [13] can be learned with a small amount of training, but only operate on highly structured documents; that is, the relations have been formatted semi-structurally and hence, the challenge is in segmenting entities, rather than mining relations. Some recent IE systems adopt bootstrapping techniques to discover tuples [2, 5, 16]. With a small set of domain-specific seed tuples (e.g., book titles and authors), the systems learn extraction patterns from them and use the learned patterns to extract more tuples, which in turn are used to learn more patterns. 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-early 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

[1] Lemur toolkit for language modeling and information retrieval. `http://www-2.cs.cmu.edu/~lemur`.

[2] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of ACM JCDL'00*, 2000.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of VLDB'94*, pages 487–499, 1994.

[4] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *SIGMOD 2001*.

[5] S. Brin. Extracting patterns and relations from the World-Wide Web. In *Proc. of the 1998 Intl. Workshop on the Web and Databases*, 1998.

[6] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Elsevier Science & Technology, 2002.

[7] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: Combining semi-markov extraction processes and data integration methods. In *Proceedings of SIGKDD'04*, 2004.

[8] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. SemTag and Seeker: Bootstrapping the semantic web via automated semantic annotation. In *WWW 2003*.

[9] R. Goldman and J. Widom. WSQ/DSQ: a practical approach for combined querying of databases and the Web. In *SIGMOD 2000*.

[10] R. Guha and R. McCool. Tap: Towards a web of data. Available at http://tap.stanford.edu/.

[11] J. Heflin and J. Hendler. Searching the web with SHOE. In *Proceedings of AAAI-2000 Workshop on AI for Web Search*, 2000.

[12] P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: Categorizing hidden web databases. In *SIGMOD 2001*.

[13] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *IJCAI 1997*.

[14] C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. In *WWW 2001*.

[15] B. Magnini, M. Negri, and H. Tanev. Is it the right answer? exploiting Web redundancy for answer validation. In *Proceedings of ACL'02*, pages 425–432, 2002.

[16] E. Riloff and R. Jones. Learning dictionaries form information extraction by multi-level bootstrapping. In *AAAI 1999*.

[17] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

[18] S. Staab, A. Maedche, and S. Handschuh. An annotation framework for the semantic web. In *Proceedings of the First Workshop on Multimedia Annotation*, 2001.

[19] P. D. Turney. Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. In *ECML 2001*.