WHIRL: A Word-based Information Representation Language

William W. Cohen
AT&T Labs–Research
Shannon Laboratory
180 Park Avenue Florham Park, NJ 07932
wcohen@research.att.com

November 20, 1998

Abstract

We describe WHIRL, an "information representation language" that synergistically combines properties of logic-based and text-based representation systems. WHIRL is a subset of non-recursive Datalog that has been extended by introducing an atomic type for textual entities, an atomic operation for computing textual similarity, and a "soft" semantics; that is, inferences in WHIRL are associated with numeric scores, and presented to the user in decreasing order by score. We show that WHIRL strictly generalizes both ranked retrieval of documents, and logical deduction; that non-trivial queries about large databases can be answered efficiently; that WHIRL can be used to accurately integrate data from heterogeneous information sources, such as those found on the Web; that WHIRL can be used effectively for inductive classification of text; and finally, that WHIRL can be used to semi-automatically generate extraction programs for structured documents.

1 Introduction

Most existing information systems adopt either a logic-based or text-based approach to representing information. The logic-based approach, exemplified by database systems and knowledge representation systems, is good for describing the relationship between objects, and the structure of complex entities. In the text-based approach, most information is stored as raw, natural-language text. This approach is good for storing less structured information about the world.

In this paper we describe WHIRL (for Word-based Heterogeneous Information Representation Language), a new type of information system that synergistically combines logic-based and text-based representation methods. With respect to text, WHIRL adopts a key tool of modern text-based information systems: the *term-weight* representation for text, in which a document is represented as a set of *terms*, each associated with a numeric *weight* indicating its relative importance. (This is sometimes called a "bag of words" representation, since terms correspond roughly to words.)

Term-based representations can be easily created and stored, and with suitable indices, many operations can be carried out very efficiently. Another compelling advantage of this representation is that with a good weighting scheme, the term-weight representation of a document can be a surprisingly effective model of its semantic content; in particular, documents with intuitively similar semantic content often have similar representations. The similarity of represention is often exploited by modern statistical information retrieval (IR) systems. For example, most modern text retrieval methods work by taking a query q, treating it as a (short) document d_q, and returning to the user the documents most similar to d_q, in decreasing order of similarity. Textual similarity is thus the basis for most ranked-retrieval IR systems.

In WHIRL, this notion of similarity has been closely integrated with logical deduction. WHIRL is a conventional logic (a subset of non-recursive Datalog) that has been extended by introducing an atomic type for textual entities, and an atomic operation for computing textual similarity. The presence of the "soft" similarity predicate necessitates a "soft" semantics; inferences in WHIRL are associated with numeric scores, and presented to the user in decreasing order by score, much like the documents returned by a ranked-retrieval IR system.

In this paper, we will survey several applications of WHIRL. We will show that WHIRL strictly generalizes both IR ranked retrieval, and logical deduction; that non-trivial queries concerning large databases can be answered efficiently; that WHIRL can be used to integrate data from distinct, distributed, heterogeneous, information sources, such as those found on the Web; that WHIRL can be used effectively for inductive classification of text; and finally, that WHIRL can be used to extract data from structured documents, and to semi-automatically generate "wrappers" (extraction programs) for structured documents.

Some of the material in this paper is a survey of results that have appeared elsewhere [6, 7, 8, 9, 10, 11]. Notationally, we will use logic-programming syntax and terminology throughout the paper; however, many of the ideas can be (and have been) expressed in database terminology and SQL syntax.

2 An Overview of WHIRL

2.1 The Underlying Representation for Text

We begin with an overview of the data model used by WHIRL. In WHIRL, the fundamental items that are manipulated by the logic are not constant values, but entities that correspond to fragments of text. We call these entities *simple texts*—"simple" emphasizing that the text is assumed to have no additional structure. For instance, in representing the information shown in Table 1, simple text entities might be used to represent the text fragments "Roberts Theater Chatham", "Brassed Off", "7:15 - 9:10", "Space Balls, 1987", and so on. Each simple text is represented internally as a *document vector* [31]. We assume a vocabulary T of *terms*; in the current implementation of WHIRL, terms are word stems (morphologically inspired prefixes) produced by the Porter stemming algorithm [29]. A simple text is represented as a

Cinema	Movie	Show Times
Roberts Theaters Chatham	Brassed Off	7:15 - 9:10
Berkeley Cinema	Hercules	2:00 - 4:15 - 7:30
Sony Mountainside Theater	Men In Black	7:40 - 8:40 - 9:30 - 10:10
⋮	⋮	⋮

Movie	Review
Men in Black, 1997	$(***)$ One of the summer's biggest hits, this ...
Face/Off, 1997	$(**\frac{1}{2})$ After a somewhat slow start, Cage and Travolta ...
Space Balls, 1987	$(*\frac{1}{2})$ While not one of Mel Brooks' better efforts, this Star Wars spoof...
⋮	⋮

Figure 1: Two tables containing text

vector of real numbers $\vec{v} \in \mathcal{R}^{|T|}$, each component of which corresponds to a term $t \in T$. We will denote the component of $\vec{v}$ which corresponds to $t \in T$ by v_t.

The general idea behind the vector representation is that the magnitude of the component v_t is related to the "importance" of the term t in the document represented by $\vec{v}$. Two generally useful heuristics are to assign higher weights to terms that are *frequent* in the document, and to terms that are *infrequent* in the collection as a whole; the latter terms often correspond to proper names and other particularly informative terms. We use the TF-IDF weighting scheme [31]: for a document vector $\vec{v}$ appearing in a collection C, we let v_t be zero if the term t does not occur in text represented by $\vec{v}$, and otherwise let $v_t = (\log(TF_{\vec{v},t}) + 1) \cdot \log(IDF_t)$. In this formula, $TF_{\vec{v},t}$ is the number of times that term t occurs in the document represented by $\vec{v}$, and $IDF_t = \frac{|C|}{n_t}$, where n_t is the total number of documents in C that contain the term t.

For instance, if $C_{2,1}$ is the collection of simple texts that occur in the first column of the second table of Figure 1, then the document vector $\vec{v}_{\mathrm{MIB}}$ for the text "Men in Black, 1997" in $C_{2,1}$ will have four non-zero components: v_{men}, v_{in}, v_{black}, and v_{1997}. The component v_{1997} will have a small value, since "1997" is a common term in the collection. Notice that the vectors are conceptually long, but very sparse, and hence can be efficiently stored and manipulated.

One advantage of this "vector space" representation is that the similarity of two documents can be easily computed. The *similarity* of two document vectors $\vec{v}$ and $\vec{w}$ is given by the formula

$$SIM(\vec{v}, \vec{w}) = \sum_{t \in T} \frac{v_t \cdot w_t}{||\vec{v}|| \cdot ||\vec{w}||}$$

This is usually interpreted as the cosine of the angle between $\vec{v}$ and $\vec{w}$. It will be large if the two vectors share many "important" terms. Notice that $SIM(\vec{v}, \vec{w})$ is always between zero and one.

For instance, if $C_{1,2}$ is the collection of simple texts that occur in the second column of the first table of Figure 1, then the document vector for the text "Men in Black" in $C_{1,2}$ will be highly similar to the vector $\vec{v}_{\mathrm{MIB}}$ described above: these simple texts differ on only one term, "1997", which has a high document frequency in $C_{2,1}$, and hence low weight. The simple texts for "Brassed Off" and "Face/Off" will have low but non-zero similarity, as they share the single term "Off", and all other pairs of movie names shown will have similarity of zero.

2.2 Semantics of Conjunctive Queries

Like a conventional deductive database program, a WHIRL program consists of two parts: an *extensional database* (EDB), and an *intensional database* (IDB). The IDB is a non-recursive set of function-free definite clauses. The EDB is a collection of ground atomic facts, each of which consists of a predicate name, a tuple of simple texts, and a numeric *score*. (Roughly speaking, the score indicates a degree of belief in this proposition.) Elsewhere, we have called this data model STIR, for Simple Texts in Relations.[1] For example, the first row of the first table in Figure 1 might be represented with an EDB fact *movieListing*$(\vec{v_1}, \vec{v_2}, \vec{v_3})$ with a score of 1, where $\vec{v_1}$ encodes "Roberts Theater Chatham", $\vec{v_2}$ encodes "Brassed Off", and $\vec{v_3}$ encodes "7:15-9:10". The first row of the second table in Figure 1 might be represented with an EDB fact *review*$(\vec{w_1}, \vec{w_2})$ with a score of 1, where $\vec{w_1}$ encodes "Men In Black, 1997" and and $\vec{w_2}$ encodes the complete text of the review: "($* * *$) One of the summer's biggest hits ..."

In formally describing the semantics of the WHIRL logic, we will first consider *flat conjunctive queries*—conjunctions of *EDB literals* and *similarity literals*. An *EDB literal* is of the form $p(X_1, \ldots, X_k)$, where the X_i's are variable symbols, and p is the name of an EDB predicate. A *similarity literal* is of the form $X \sim Y$, where X and Y are variable symbols. (The operator "$\sim$" should be read "is similar to") We will assume that all variables in a conjunctive query appear at least once in some EDB literal.

In Datalog, the answer to a conjunctive query would be the set of ground substitutions that make the query "true" (*i.e.*, provable against the EDB). In WHIRL, the notion of provability will be replaced with a "soft" notion of *score*: substitutions with a higher score will be ranked higher in the list of answers shown to the user.

Specifically, given an EDB, we define the *score* of a ground substitution θ for a literal B as follows. If B is an EDB literal $p(X_1, \ldots, X_a)$, then $SCORE(B, \theta) = s$ if $B\theta$ is a fact in the EDB with score s, and $SCORE(B, \theta) = 0$ otherwise. If B is a similarity literal $X \sim Y$, then $SCORE(B, \theta) = SIM(\vec{x}, \vec{y})$, where $\vec{x} = X\theta$ and $\vec{y} = Y\theta$; in other words, $SCORE(W, \theta)$ is the cosine similarity between the documents represented by $X\theta$ and $Y\theta$. The *score* of a substitution for a flat conjunctive query $Q = B_1 \wedge \ldots \wedge B_k$ is defined to be the product of the scores of the conjuncts:

$$SCORE(B_1 \wedge \ldots \wedge B_k, \theta) = \prod_{i=1}^{k} SCORE(B_i, \theta) \tag{1}$$

[1] Note that the deductive database formalism considered here, without recursion, is closely related to relational databases.

A *full answer* to a flat conjunctive query Q is an ordered list of all substitutions θ with positive score, presented in non-increasing order by score.

Allowing for a moment the syntactic sugar of allowing constants to appear in queries, the following is an example of a flat conjunctive query:

(Q_1) ?- *review(Movie,Review)* $\wedge$ *Review*$\sim$ *"comedy with space aliens"*

The full answer to this query is a list of all substitutions that bind *Review* to some document vector somewhat similar to the simple text "comedy with space aliens", ordered in decreasing order of similarity. This is exactly the result that would be obtained if one used a IR ranked-retrieval system to search a corpus of movie reviews; hence WHIRL generalizes IR ranked retrieval.

As another example, the query

(Q_2) ?- *movieListing(Cinema,Movie1,Times)* $\wedge$ *review(Movie2,Review)* $\wedge$ *Movie1*$\sim$*Movie2*

associates a review with a movie listing if the two movie names involved are similar. For instance, it would associate the listing for "Men in Black" with the review for "Men in Black, 1997", and the listing for "Brassed Off" with the review for "Face/Off". The first of these inferences would have a higher score, since the names involved are more highly similar. More precisely, the full answer to this query would be a list of bindings for the variables *Cinema, Movie1, Times, Movie2*, and *Review*, with the bindings that make *Movie1* and *Movie2* most similar presented first.

Readers familiar with databases will recognize the clause above as a "soft" variant of the natural join of the *movieListing* and *review* relations:

(Q_3) ?- *movieListing(Cinema,Movie,Times)* $\wedge$ *review(Movie,Review).*

The "hard" query, Q_3, is appropriate when the *Movie* fields of the tables contain clear, unambiguous identifiers—that is, when each movie is consistently associated with a single, unique string (like "*movie123*"). However, unambiguous identifiers are not likely to exist if data is taken from different sources. For instance, if the two tables shown in Figure 1 were extracted from two different Web sites, it would be unrealistic to assume that the names used to identify movies are perfectly consistent across both tables. The "soft" version of the join, Q_2, makes a much weaker assumption—namely, that names for the same object will be similar.

A more complex query might search reviews by plot to where the latest science fiction comedy is playing at the Mountainside Theater:

(Q_4) ?- *movieListing(Cinema,Movie1,Times)*
$\wedge$ *review(Movie2,Review)*
$\wedge$ *Movie1*$\sim$*Movie2*
$\wedge$ *Review*$\sim$ *"comedy with space aliens"*

Query Q_4 combines IR ranked retrieval with a soft join.

2.3 Definite Programs and Universal Quantification

To extend these semantics to definite programs, we adopt a dual version of Equation 1 for scoring disjunctions:

$$SCORE(Q_1 \vee \ldots \vee Q_k, \theta) = 1 - \prod_{i=1}^{k}(1 - SCORE(Q_i, \theta)) \tag{2}$$

Following database terminology, a WHIRL predicate is called a *view*. Consider a "flat" view definition, in which each clause body is a flat conjunctive query:

$v(X_1, \ldots, X_k) \leftarrow Body_1.$
$\ldots$
$v(X_1, \ldots, X_k) \leftarrow Body_k.$

Since we consider only non-recursive Datalog, views that are not flat can be "unfolded" by repeated resolution against the IDB until they contain only EDB literals; henceforth we will assume views are flat. For any possible atom $a = v(a_1, \ldots, a_k)$ that instantiates the head of this clause, define the *support* of a, $SUPPORT(a)$, to be the set of all pairs $\langle\theta, Body_i\rangle$ such that $v(X_1, \ldots, X_k)\theta = a$, $Body_i\theta$ is ground, and $SCORE(Body_i, \theta) > 0$. Following Equation 2 and the usual semantics of logic programs, we define the score of a to be

$$1 - \prod_{\langle\theta, Body_i\rangle \in SUPPORT(a)} (1 - SCORE(Body_i, \theta)) \tag{3}$$

Notice that in most cases, only high-scoring answers will be of interest. Therefore, the operations usually performed by a user (or program) interacting with WHIRL are to define views, and then *r-materialize* them. To r-materialize a view, the interpreter finds the r highest-scoring ground atoms a associated with the view, and stores these facts in the EDB for later use. Here r is a parameter supplied by the user (or calling program). More detail on the semantics of WHIRL can be found elsewhere [7, 8].

A recent extension to the WHIRL logic is a soft version of universal quantification. Currently this operator is written *many(Template,Test)* where the *Test* is an ordinary conjunction of literals, and the *Template* is a single literal of the form $p(Y_1, \ldots, Y_n)$, where p is an EDB predicate and the Y_i's are all distinct; also, the Y_i's may appear only in *Test*. As an example, to ask for movies that won many academy awards, one might use the query

movieListing(C,M,T) $\wedge$ many(academy_award(Y),winner(M,Y)).

The score of a "many" clause is the weighted average score of the *Test* conjunction on items that match the *Template*. Formally, for a substitution θ and a conjunction W,

$$SCORE(many(p(Y_1, \ldots, Y_k), Test), \theta) = \tag{4}$$
$$\sum_{\langle s, a_1, \ldots, a_k\rangle \in P} \frac{s}{S} \cdot SCORE(Test, (\theta \circ \{Y_1 = a_1, \ldots, Y_k = a_k\})) \tag{5}$$

where P is the set of all tuples $\langle s, a_1, \ldots, a_k \rangle$ such that $p(a_1, \ldots, a_k)$ is a fact in the EDB with score s, and $S = \sum_{\langle s,a_1,\ldots,a_k\rangle \in P} s$ is the sum of all such scores s. When the score of every EDB fact concerning the predicate p/k has score 1, then Equation 4 can be simplified to

$$SCORE(many(p(Y_1, \ldots, Y_k), Test), \theta) =$$
$$\frac{1}{|P|} \sum_{\langle s,a_1,\ldots,a_k\rangle \in P} SCORE(Test, (\theta \circ \{Y_1 = a_1, \ldots, Y_k = a_k\}))$$

One can interpret this as the expected score of $Test\theta\sigma$, where σ binds the template to a randomly chosen matching fact from the EDB.

The current implementation of this construct is limited, in that only one *many* construct is allowed in a clause body. This limitation may be relaxed in the future.

3 Efficiency of WHIRL

WHIRL could be implemented very easily if efficiency were not an issue; for instance, a Prolog meta-interpreter with chronological backtracking could be used to derive all atoms with non-zero score, which could then be collected and sorted by score. Unfortunately, a naive implementation of WHIRL is likely to be extremely slow.

One issue is that in most practical problems, many simple texts will be weakly similar. For example, consider computing the "soft intersection" of two large (size n) lists of company names with the query $list1(X) \wedge list1(Y) \wedge X \sim Y$. Because any pair of names that both contain the term "inc" would have non-zero similarity, the full answer to this query (*i.e.*, the set of all substitutions with non-zero score) will be large. Indeed, if we assume that a fixed fraction $\frac{1}{k}$ of company names contain the term "inc", then one would expect the full answer to contain around $(\frac{n}{k})^2$ positive-scoring substitutions, simply due to the matches on the term "inc". Further, the size of the full answer for the soft intersection of m lists of this sort would be around $(\frac{n}{k})^m$. Similar results occur in any natural domains. For instance, in intersecting two lists of movie identifiers of the type used in the second table of Figure 1 (*e.g.*, "Men in Black, 1997"), there would be many weak matches on terms corresponding to years.

These properties of the term-based similarity metric mean that computing full answer sets will necessaily be very inefficient. For large problems, it is not even practical to store all the non-zero pairwise similarity scores, since this will typically be quadratic in the size of the original EDB.

The notion of r-materialization is intended to circumvent this problem: since there are usually too many possible inferences, we limit the goal of deduction to finding a few good inferences. WHIRL's inference algorithms are designed to find a few good answers to a query, without generating all possible answers. More specifically, the fundamental inference step in WHIRL is to find the r highest-scoring substitutions for a conjunctive query. The algorithm for accomplishing these steps efficiently is based on a combination of admissible A$*$ search with IR indexing and pruning methods [8, 7]. Higher-level operations associated with the logic can be implemented or approximated using this operation as a subroutine.

These design decisions lead to a fairly efficient inference scheme, as demonstrated by experiments reported elsewhere [7, 8]. Below, we will summarize one such result. In this

	Domain	
	Games	Birds
# sites indexed	15	34
# facts stored in EDB	23,435	143,666
# queries in sample	100	91
avg time/query (sec)	0.3	0.2
max time/query (sec)	5.2	5.4

Table 1: Performance of the WHIRL interpreter on real-world queries.

experiment, WHIRL was used as the central component of a working, Web-based, *information integration* system [9, 10]. This system downloads data from a number of related Web sites and extracts a WHIRL knowledge base, which can then be queried. Two domains were implemented for this system, one integrating information on birds of North America, and one integrating information about educational computer games. Both domains were made available on the Web for use by the general public, and queries issued to the system were logged.

Later, we took a "snapshot" of each domain[2], and measured the response time for a subset of these queries. In the game domain, we took a random sample of 100 queries. In the bird domain, which includes a browsing interface and a "quick search" feature, we took all queries (over a period of several days) which used the "advanced search" feature, thus excluding many of the simpler queries; there are 91 queries in this sample. Table 1 provides a summary of the results. The average response time is well under a second, even for these real-world queries submitted against moderately large databases.

4 Accuracy of WHIRL

4.1 Accuracy of Integration Inferences

Efficient reasoning is only worthwhile if the inferences made are useful ones. Therefore, we also evaluated the *accuracy* of similarity joins, again using data taken from the Web. We selected pairs of Web-derived EDB predicates which contained two or more plausible "key" fields. As an example, we might use two predicates *p(Name,Homepage)* and *q(Name,Homepage)*, pairing the name of an company and its homepage URL. One of these fields, the "primary key", was used in the similarity literal in the join. The second key field was then used to check the correctness of proposed pairings; specifically, a pairing was marked as "correct" if the secondary keys matched (using an appropriate matching procedure) and "incorrect" otherwise. For the predicates p and q above, for example, it would be appropriate to use the *Name* field as the primary key, and the *Homepage* field as the secondary key.

[2]Note that the size of the databases change every time data is downloaded from the Web.

	Domain	Size	Average Precision	Notes
1	Movies	78 × 421	100.0%	
2	Movies	78 × 421	98.8%	[1]
3	Movies	78 × 421	94.6%	[1]
4	Animals	11 × 52	100.0%	
5	Animals	23 × 27	99.6%	
6	Animals	16 × 78	97.1%	
7	Animals	990 × 4719	95.2%	
8	Birds	15 × 23	100.0%	
9	Birds	155 × 564	99.1%	
10	Birds	20 × 317	91.4%	
11	Birds	68 × 914	95.8%	
12	Birds	68 × 914	83.0%	[1]
13	Businesses	532 × 331	99.5%	[3]
14	Businesses	1163 × 976	84.9%	[2]
15	National Parks	398 × 258	95.7%	
16	Computer Games	113 × 1626	86.1%	
17	Computer Games	75 × 1626	67.1%	[1]

Table 2: Accuracy of similarity joins. Notes: [1] In these problems, names were not completely extracted from the surrounding text. [2] On this problem, the secondary keys have a substantial number of errors. [3] Items being joined are descriptions containing name, address, and phone number.

We then treated correct pairings in the same way that "relevant" documents are typically treated in evaluation of a ranking proposed by a standard IR system. In particular, we measured the quality of a ranking using *(non-interpolated) average precision*. To motivate this measurement, assume the end user will scan down the list of answers and stop at some particular "target answer" that he or she finds to be of interest. The answers listed below this "target" are not relevant, since they are not examined by the user. Above the target, one would like to have a high density of correct pairings; specifically, one would like the set S of answers above the target to have high *precision*, where the precision of S is the ratio of the number of correct answers in S to the number of total answers in S. *Average precision* is the average precision for all "plausible" target answers, where an answer is considered a plausible target only if it is correct. To summarize, letting a_k be the number of correct answers in the first k, and letting $c(k) = 1$ iff the k-th answer is correct and letting $c(k) = 0$ otherwise, average precision is the quantity $\sum_{k=1}^{r} c(k) \cdot \frac{a_k}{k}$.

In the experiments, we used the 1000 top-scoring pairings to compute average precision. Note that average precision is 1 only when all correct answers precede all incorrect answers. The results for several such pairs of predicates are summarized in Table 2. The table also indicates the size of the tables being joined: for example, a size of "990 × 4719" means that the first table has 990 tuples, and the second has 4719.

In most of the results from the table, the items being joined are names of objects in appropriate domain: for instance, the items being joined are movie names in row 1, and the names of animals or birds in rows 4-10. In practise, however, it is not always possible to extract names from the surrounding text. Therefore, we also conducted some controlled experiments in which similarity joins were applied to names that not fully extracted from the surrounding text. In line 2, we joined movie names to the items containing the full text of a movie review; note that the reviews always contain the name of the movie being reviewed, but also a great deal of additional text. In line 3, we joined movie reviews with "movie listings", each containing a movie name, cinema name and address, and show times. In line 12, we joined bird names with paragraph-length descriptions of sound files.[3] In line 17, we joined game names with sentence- to paragraph- length sections of text that describe where to obtain on-line game demos. Examples of these can be found in Table 3, which presents the first few pairings in this join. (The check marks and crosses in the final column of the table indicate if a pairing was considered correct. This table also provides a concrete illustration of average precision. The average precision for this ranking is 67.1%—the *worst* average precision result from Table 2.)

On these domains, similarity joins are extremely accurate, relative to the secondary keys. The average value for average precision across the rows involving extracted data is over 97%. Furthermore, using incompletely extracted names often leads to only a small degradation in average precision. The experiments of this section are reported in more detail elsewhere [7, 8].

4.2 Average precision for restricted similarity joins

In another set of experiments, we looked at *restricted similarity joins*—queries of the form

$$p(X_1, \ldots, X_i, \ldots, X_a) \wedge q(Y_1, \ldots, Y_j, \ldots, Y_b) \wedge X_i \sim Y_j \wedge X_k \sim d$$

In this case the similarity join is restricted to tuples for which some field is similar to d, where d is some simple text document specified by the user. This is analogous to imposing an additional selection criterion on an ordinary relational join; alternatively, one can think of these queries as combining a similarity join with an IR-type ranked retrieval. Query Q_4 (on page 5) is an example of a restricted similarity join.

We evaluated these types of queries in two domains—the movie domain, and the bird domain. In the movie domain, we evaluated the performance of several variations of Q_4, each variation replacing the text "comedy with space aliens" with a different text d. We used for d five plot descriptions, each intended to pick out a single movie.[4] The plot descriptions were provided in a "long" version (for example, "depressing czech movie set in russia", "jim carey comedy about a lawyer") and also in a "short" version (for example, "serious czech movie",

[3]For example: "Scarlet Tanager (58kB) *Piranga olivacea.* New Paltz, June 1997. '...Robin-like but hoarse (suggesting a Robin with a sore throat).' (Peterson) '..a double-tone which can only be imitated by strongly humming and whistling at the same time.' (Mathews)".

[4]This was feasible because the relations involved in the join contained only first-run movies.

Score	X_i	Y_j	
80.26	Ubi Software has a demo of Amazing Learning Games with Rayman.	Amazing Learning Games with Rayman	$\surd$
78.25	Interplay has a demo of Mario Teaches Typing. (PC)	Mario Teaches Typing	$\surd$
75.91	Warner Active has a small interactive demo for Where's Waldo at the Circus and Where's Waldo? Exploring Geography (Mac and Win)	Where's Waldo? Exploring Geography	$\surd$
74.94	MacPlay has demos of Marios Game Gallery and Mario Teaches Typing. (Mac)	Mario Teaches Typing	$\surd$
71.56	Interplay has a demo of Mario Teaches Typing. (PC)	Mario Teaches Typing 2	$\times$
68.54	MacPlay has demos of Marios Game Gallery and Mario Teaches Typing. (Mac)	Mario Teaches Typing 2	$\times$
68.45	Psygnosis has an interactive demo for Lemmings Paintball. (Win95)	Lemmings Paintball	$\surd$
65.70	ICONOS has a demo of What's The Secret? Volume 1. (Mac and Win)	What's the Secret?	$\surd$
64.33	Fox Interactive has a fully working demo version of the Simpsons Cartoon Studio. (Win and Mac)	Simpsons Cartoon Studio	$\surd$
62.90	Gryphon Software has demos of Gryphon Bricks, Colorforms Computer Fun Set - Power Rangers and Sailor Moon, and a FREE Gryphon Bricks Screen Saver (Mac and Win).	Gryphon Bricks	$\surd$
60.30	Vividus Software has a free 30 day demo of Web Workshop (web authoring package for kids!) Win 95 and Mac	Web Workshop	$\surd$
59.96	Conexus has two shockwave demos - Bubbleoids (from Super Radio Addition with Mike and Spike) and Hopper (from Phonics Adventure with Sing Along Sam).	Super Radio Addition with Mike & Spike	$\surd$
59.84	IBM and Crayola have demos of Magic Wardrobe (Mac and Win) and Magic 3-D Coloring Book (Win)	Crayola Magic 3D Coloring Book	$\surd$
56.42	Gryphon Software has demos of Gryphon Bricks, Colorforms Computer Fun Set - Power Rangers and Sailor Moon, and a FREE Gryphon Bricks Screen Saver (Mac and Win).	Sailor Moon Colorforms Computer Fun Set	$\surd$
56.24	Gryphon Software has demos of Gryphon Bricks, Colorforms Computer Fun Set - Power Rangers and Sailor Moon, and a FREE Gryphon Bricks Screen Saver (Mac and Win).	Power Rangers Colorforms Computer Fun Set	$\surd$
55.54	Simon & Schuster Interactive brings the 3-D Action Learning line, Chill Manor, Spaceship Pythagoras, and I.M. Meen. They're currently offering a demo of I.M. Meen (DOS).	I.M. Meen	$\surd$

Table 3: Result of a similarity join involving partially extracted data in the domain of computer games. A pairing is judged correct if the text X_i contains a reference to the game Y_j.

Domain	Restriction	Average Precision
Movies	long plot 1	94.4%
	long plot 2	72.5%
	long plot 3	76.3%
	long plot 4	98.2%
	long plot 5	100.0%
Movies	short plot 1	100.0%
	short plot 2	66.3%
	short plot 3	75.1%
	short plot 4	100.0%
	short plot 5	100.0%
Bird Calls	call 1	100.0%
	call 2	20.0%
	call 3	100.0%
	call 4	100.0%
	call 5	100.0%

Table 4: Average precision for restricted similarity joins in several domains.

"lawyer comedy"). An answer was judged correct if the pairing of film names matched the secondary key, and involved the correct movie.

In the bird domain, we applied a similar methodology. We associated the name of a bird a paragraph-length description of an appropriate sound file, thus generating a predicate *birdcall(Name, Sound)*. We took a second list of bird names *birdlist(Name)* and evaluated the five versions of the query

?- *birdlist(Name1)*
 $\wedge$ *birdcall(Name2,Sound)*
 $\wedge$ *Name1*$\sim$*Name2*
 $\wedge$ *Sound*$\sim$ *d*

For *d*, we used five short sound descriptions (for example, "an owl hooting").

The results are shown in Table 4. The average precision for these WHIRL queries is lower, but still quite good—usually between 66% and 100%, and averaging around 86%.

More preliminary experiments with WHIRL on the same sort of problem have been reported elsewhere [6].

4.3 Accuracy of Classification Inferences

Many of the sources used in the integration applications are incomplete. This is particularly true in the games domain, which changes rapidly as new games are introduced. We observed that soft, similarity based queries to such sources often give reasonable answers even when the information sources are apparently inadequate.

For instance, in the game domain there are lists of reviews, encoded as EDB facts like *review("Putt-Putt Travels Through Time", "http://gamereview.com/r/pptime.html")*.

There are also sources that associate game titles with broad categories, leading to EDB facts like *category("Putt-Putt's Fun Pack", "adventure")* and *category("Time Traveler CD", "history")*. To find reviews of adventure games one might use the view

v(Game1,Url) $\leftarrow$
  review(Game1,Url) $\wedge$ *category(Game2,Cat)*
  $\wedge$ *Game1*$\sim$*Game2* $\wedge$ *Cat*$\sim$ *"adventure"*

The soft match between the game titles (*Game1*$\sim$*Game2*) is needed because the titles might vary in the *review* and *category* predicates. One side effect of using the soft match, however, is that the query may return reviews for games that are *not* included in the *category* predicate. For instance, with an EDB containing only facts given above, the review of "Putt-Putt Travels Through Time" will be (tentatively) considered a review of an *adventure* game, because the name of this game is similar to the name of "Putt-Putt's Fun Pack". The same review might also be suggested as a review of a *history* game, albeit with somewhat lower confidence, due to the similarity with the game title "Time Traveler CD". In both cases, reasonable inferences are being made by WHIRL on the basis of incomplete information.

Motivated by these observations, we also evaluated the accuracy of a second type of query—one closely related to the task of inductive classification. Suppose one is given several facts of the form *train(inst,lab)* associating an "instance" *inst* with some "label" *lab*, where the label is taken from a relatively small fixed set of possible labels. One can assign a label to a new instance by first adding the fact *test(inst)* to the EDB, and then materializing the view

v(X,Lab) $\leftarrow$ *test(X)* $\wedge$ *train(Y,Lab)* $\wedge$ $X\sim Y$.

This query associates labels with X based on X's similarity to instances Y in the training data. Used in this manner, WHIRL is a sort of nearest-neighbour classifier. Algorithmically, this use of WHIRL is similar to a vector space distance-weighted K-NN method investigated by Yang and Chute [34]. However, WHIRL uses a different scheme to combine the weights associated with the K closest neighbours of an unclassified instance; Yang and Chute's method uses the sum of similarities for all neighbors with label *Lab* as the combined evidence for label *Lab*, and WHIRL uses Equation 2 to combine evidence.

To evaluate thus use of WHIRL, we created nine benchmark problems using data found on the World Wide Web, all of which associate short, name-like strings with labels from a finite set. We compared the generalization accuracy of WHIRL to the accuracy of several other inductive learning methods, including Yang and Chute's method (K-NN); 1-NN, using cosine distance as the distance function for instances; C4.5 [30] using a binary representation; and Ripper [4] using set-valued features [5]. We used $r = 30$ in WHIRL's r-materialization step, and $K = 30$ for Yang and Chute's method. Minimal feature selection was used for C4.5 (terms appearing in less than three examples were discarded) and no feature selection was used for the remaining learning methods. Standard experimental methodology (10-fold cross-validation, or a held-out test set) was used to estimate generalization error.

The results are shown in Table 5. The first three columns indicate the number of training examples, test examples, and classes in each dataset; the remaining columns indicate the accuracies of the various learning systems. WHIRL outperforms 1-NN and C4.5 eight of nine

	Problem Characteristics			Percent Accuracy				
	#train	#test	#classes	1-NN	C4.5	RIPPER	K-NN	WHIRL
memos	334	10cv	11	68.9	57.5	50.9	64.4	66.5
cdrom	798	10cv	6	40.5	39.2	38.3	47.2	47.5
birdcom	914	10cv	22	70.4	79.6	88.8	82.3	83.9
birdsci	914	10cv	22	82.8	83.3	91.0	89.2	90.3
hcoarse	1875	600	126	20.2	30.2	28.0	32.7	32.9
hfine	1875	600	228	12.7	17.2	16.5	21.0	20.8
species	3119	1600	6	53.0	89.4	90.6	93.6	94.3
books	3501	1800	63	87.5	52.2	42.3	60.1	61.4
netvet	3596	2000	14	50.9	68.8	67.1	67.8	68.1
average				54.10	57.41	57.52	62.00	62.90

Table 5: Accuracy of WHIRL used for inductive classification

times, and outperforms RIPPER seven of nine times. The competing method that is closest in performance to WHIRL is Yang and Chute's K-NN method. However, WHIRL also outperforms Yang and Chute's method eight of nine times, and although all of these eight differences are small, five of them are statistically significant. WHIRL is never significantly worse than any other learner on any individual problem.

The comparision with 1-NN is of special interest, because for queries of this type, 1-NN is equivalent to scoring a disjunction with the rule

$$SCORE(Q_1 \vee \ldots \vee Q_k, \theta) = \max_i(SCORE(Q_i, \theta)) \tag{6}$$

instead of Equation 2. In the database community, it is frequently proposed that Equation 6 be used as the combination rule for disjunction, and its dual (using *min* instead of *max*) be used for conjunction. However, this combination rule is empirically inferior to Equation 2 on problems of this sort.

Further details on these experiments can be found elsewhere [11].

5 Extraction from structured documents using WHIRL

5.1 WHIRL as a Wrapper Language

One bottleneck in implementing knowledge integration systems is developing *wrappers*—programs that extract data from Web pages. As an illustration of this, Figure 2 shows the HTML encoding of two Web pages, together with the data that should be extracted from each. Because data can be presented in many different ways on the Web, and because Web pages frequently change format, building and maintaining these wrappers is often time-consuming and tedious.

In past work, we used a special-purpose language for writing wrappers [9, 10]. A program in this language manipulates an HTML parse tree (that is, a tree with nodes labeled by tag

HTML Source for Page 1:

```
<html><head>...</head>
<body>
<h1>Editorial Board Members</h1>
<table> <tr>
    <td>G. R. Emlin, Lucent</td>
    <td>Harry Q. Bovik, Cranberry U</td>
  </tr> <tr>
    <td>Bat Gangley, UC/Bovine</td>
    <td>Pheobe L. Mind, Lough Tech</td>
  </tr> <tr>
...
</table>
...
```

Extracted data:

G. R. Emlin, Lucent
Harry Q. Bovik, Cranberry U
Bat Gangly, UC/Bovine
...

HTML Source for Page 2:

```
<html><head>...</head>
<body>
<h1>My Publications</h1>
<ul>
<li>Optimization of fuzzy neural
  networks using distributed parallel
  case-based genetic knowledge discovery
  (<a href="buzz.ps">postscript</a>,
   <a href="buzz.pdf>PDF</a>)</li>
<li>A linear-time version of GSAT
  (<a href="peqnp.ps">postscript</a>)</li>
...
```

Extracted data:

Optimization ... (postscript,PDF)	buzz.ps
Optimization ... (postscript, PDF)	buzz.pdf
A linear-time version of ...	peqnp.ps
...	...

Figure 2: Two sample HTML pages, and the data that would be extracted from each.

names like *body*, *table*, and *ul*), primarily by deleting and relabeling nodes. A wrapper in this language converts the parse tree for a Web page into another tree labeled with terms from the knowledge base; the relabeled tree can then be stored directly in the knowledge base. This language is clearly not sufficient for all purposes, since it is impossible to extract along boundaries not indicated by HTML markup commands; for instance, it is impossible to separate names from affiliations in Page 1 of Figure 2. However, the language is almost always expressive enough to build wrappers for WHIRL—in part because of WHIRL's flexibility in handling incompletely extracted data.

Recently, we have begun exploring using WHIRL itself as a wrapper implementation language. To make this possible, we begin by translating an HTML parse tree into the following EDB predicates.

- *elt(Id,Tag,Text,Position)* is true if *Id* is the identifier for a parse tree node, n, *Tag* is the HTML tag associated with n, *Text* is all of the text appearing in the subtree rooted at n, and *Position* is the sequence of tags encountered in traversing the path from the root to n. For instance, the second list item (*li*) node in Page 2 of Figure 2 might give rise to the EDB fact *elt("node_17", "li", "A linear-time version of GSAT (postscript)", "html–body–ul–li")*.

- *attr(Id,AName,AValue)* is true if *Id* is the identifier for node n, *AName* is the name of an HTML attribute associated with n, and *AValue* is the value of that attribute. For instance, if *node_18* identifies the anchor embedded in *node_17* above, then the EDB would include the fact *attr("node_18", "href", "peqnp.ps")*.

- *path(FromId, ToId, Tags)* is true if *Tags* is the sequence of HTML tags encountered on the path between nodes *FromId* and *ToId*. This path includes both endpoints, and is defined if *FromId=ToId*. For example, the path between *node_17* and *node_18* would be associated with the fact *path("node_17", "node_18", "li–a")*, and the path from *node_18* to itself would be associated with the fact *path("node_18", "node_18", "a")*.

Using these predicates, many simple extraction programs can be written quite concisely as views. For example, wrappers for the pages in Figure 2 can be written as follows.

page1(NameAffil) ←
    elt(_, _, NameAffil, "html–body–table–tr–td").
page2(Title, Url) ←
    elt(ContextElt, _, Title, "html–body–ul–li")
    ∧ *path(ContextElt, AnchorElt, "li–a")*
    ∧ *attr(AnchorElt, "href", Url).*

This "wrapper language" is incomparable in power to the one we originally implemented, but appears to be adequate in many cases. As part of our study of the extraction problem, we collected 111 different wrapper programs, all written as part of the WHIRL project, but written over several months in several different domains. This was as complete a sample of working conversion programs as we could assemble; the only existing wrappers that were discarded were ones for which no correctly-wrapped HTML pages were available, due to changes in the associated Web sites. Of these 111 wrappers, more than 80 could be easily implemented with the technique described above. Extensions to the EDB representation of the parse tree would make it possible to implement all but a handful of the wrappers.

5.2 Semi-automatic Wrapper Generation With WHIRL

A major advantage of using WHIRL views for extraction from HTML is that one can then use WHIRL to analyze the behavior of programs, and even generate programs in certain restricted classes. As an example, consider the *page2* view defined above. Many hotlist-like pages can be handled with some variant of this view, in which the tag sequences "*html–body–ul–li*" and "*li–a*" are replaced with different constants.

Let us call "*html–body–ul–li*" the *context path* for this view, and "*li–a*" the *anchor path*. The pair (*context path, anchor path*) defines a variant of the view *page2*, and thus a wrapper. Given a new HTML page known to be wrappable with a variant of the *page2* view, we could use the following view to find all possible pairs[5] of path constants:

possible_wrapper(ContextPath, AnchorPath) ←
    elt(ContextElt, _, _, ContextPath)
    ∧ *elt(AnchorElt, _, a, _)*
    ∧ *attr(AnchorElt, "href", _)*
    ∧ *path(ContextElt, AnchorElt, AnchorPath).*

[5]Note that the number of such pairs is at most quadratic in the number of nodes in the parse tree.

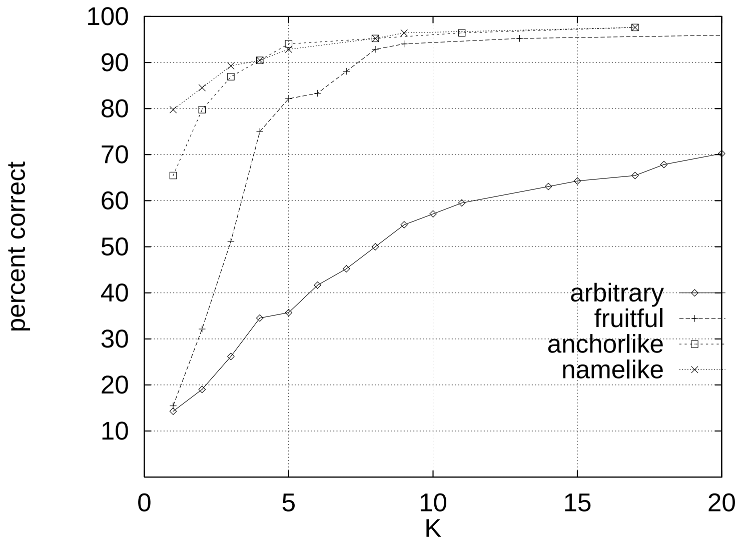

Figure 3: Semi-automatic generation of wrappers. For each ranking heuristic, the percentage of pages correctly wrapped by programs ranked in the top K is recorded, as a function of K.

The view below determines which items are extracted by a particular *ContextPath*, *AnchorPath* "program", and hence acts as a sort of interpreter for this class of wrappers:

extracted_by(ContextPath,AnchorPath,ContextElt,AnchorElt) ←
  elt(ContextElt, _, _, ContextPath) ∧ *path(ContextElt, AnchorElt, AnchorPath).*

Finally, we can use WHIRL's soft universal quantification to heuristically rank the possible wrapper programs by various criteria. One simple heuristic ranking procedures is the following:

fruitful_wrapper(CPath,APath) ←
  possible_wrapper(CPath,PPath) ∧
  many(extracted_by(CPath1,APath1,CElt,AElt),
    (CPath=CPath1 ∧ *APath1=APath)).*

This view ranks possible programs by the number of elements they extract—a trivial but useful heuristic in determining which of several repeated structures should be used in a wrapper.

Unlike the example shown in the figure, most hotlists also have the property that the text associated with the anchor is a reasonable description of the associated object. (In fact, in most wrappers of this form, the *ContextElt* and the *AnchorElt* are identical.) A useful variant of the *fruitful_wrapper* view includes as an additional heuristic a "soft" requirement that the text associated with the context element be similar to the text associated with the anchor element:

anchorlike_wrapper(CPath,APath) ←

possible_wrapper(CPath,APath) $\wedge$
many(extracted_by(CPath1,APath1,CElt,AElt),
(CPath=CPath1 $\wedge$ *APath1=APath*
$\wedge$ *elt(AElt,_,AText,_)* $\wedge$ *elt(CElt,_,CText,_)* $\wedge$ *Atext*$\sim$*CText).*

A third heuristic ranking scheme can be used when there is a known list of objects that are similar to something that might be extracted. For instance, in extracting paper titles, one might begin with a large bibliography database that overlaps with the page being extracted; in extracting movie names, one might begin with a large list of popular movies. Assuming that this information is stored in the predicate *likely_name(X)*, the *fruitful_wrapper* view can be modified to find programs that extract many items with descriptions that are similar to a "likely name":

namelike_wrapper(CPath,APath) $\leftarrow$
possible_wrapper(CPath,PPath) $\wedge$
many(name_extracted_by(CPath1,APath1,_,_),
CPath=CPath1 $\wedge$ *APath1=APath).*
name_extracted_by(ContextPath,AnchorPath,ContextElt,AnchorElt) $\leftarrow$
elt(ContextElt, _, ContextText, ContextPath)
$\wedge$ *path(ContextElt, AnchorElt, AnchorPath)*
$\wedge$ *likely_name(X)* $\wedge$ *ContextText*$\sim$*X.*

These sort of ranking schemes could be used to semi-automatically generate wrappers. Given a Web page, WHIRL could generate all possible wrapper programs from a limited class, and then allow the user to select, from the ranked list, which program should be used for this page. If a ranking lists the correct program early on, then this process would be efficient (from the users's point of view.)

To evaluate these ranking schemes, we took the initial set of 111 wrapper programs, and paired each with a single Web page that was correctly wrapped. We then selected 84 wrapper programs that could be implemented by programs in our the restricted class described above (or a similar class of wrapper programs based on the view for *page1*). For each ranking scheme described above, and each value of K from 1 to 20, we recorded the number of times that all correct wrapper programs for a Web page were listed in the top K programs in the ranking. (A few Web pages are associated with more than one wrapper program.)

For the heuristic ranking method based on "likely names", we always used lists of likely names obtained from other Web sites, if one was available. Neither of the nontrivial ranking schemes apply to all problems. In evaluating the "anchorlike wrapper" heuristic, we reverted to the "fruitful wrapper" heuristic when if was not applicable (*i.e.*, if the extracted data contained no URLs, as is the case with the *page1* view). In evaluating the "namelike wrapper" heuristic, we reverted to the "anchorlike wrapper" heuristic if no names were available, and the "fruitful wrapper" heuristic if no URLs were extracted. All the heuristic ranking methods require r-materialization of an intermediate relationship (*extracted_by* or *name_extracted_by*). In the experiments experiments we set r so that the lowest scores were around 1/2 in materializing *name_extracted_by*, and set $r = 100,000$ in materializing *extracted_by*.

The results are shown in the graph of Figure 3, the y axis here indicating the percentage of the 84 wrappers that could be correctly wrapped. These results are extremely encouraging. With the first nontrivial heuristic ranking method ("anchorlike" on the graph), about two-thirds of the wrapper programs can be correctly wrapped by the first ranked program, and thus can be implemented with essentially no work on the part of the user. With the second heuristic ranking method ("namelike"), this fraction increases to about four-fifths. While more information is required in this case (in the form of the list of likely names) this information is likely to be available when wrapping most sites. With both non-trivial heuristics, almost all pages can be wrapped using only the top 5 or so programs, and both heuristics clearly outperform the simple frequency-based ranking scheme ("fruitful" on the graph), or randomly ordering programs ("arbitrary").

To summarize, this seems to be a very promising approach to wrapper generation. With respect to human effort, it is probably more efficient than approaches based on wrapper learning [22, 19, 27], which require a human to hand-label several examples for each wrapper.

6 Conclusions

Most existing information systems adopt either a logic-based or text-based approach to representing information. WHIRL is an "information representation language" that combines elements of both of these paradigms. WHIRL is a subset of non-recursive Datalog that has been extended by introducing an atomic type for textual entities, an atomic operation for computing textual similarity, using robust and efficient term-based similarity metrics developed in the IR community. WHIRL also has a "soft" semantics; *i.e.*, inferences are associated with numeric scores, based on the degree to which similarity constraints have been satisfied, and inferences are presented to the user in decreasing order by score.

We demonstrated that WHIRL strictly generalizes both IR ranked retrieval, and logical deduction, in the sense that certain WHIRL queries can be used to emulate the actions of either an IR ranked-retrieval system or non-recursive Datalog. We showed that WHIRL can be used to integrate information extracted from multiple sites on the Web; in particular, "similarity joins" can be used to combine information from multiple sites under relatively weak assumptions. When used for this purpose, inference in WHIRL is efficient—the average time to answer queries is well under a second for real-world queries to databases containing tens of thousands of facts collected from the Web.

We also demonstrated that WHIRL is robust in several senses. It can draw accurate inferences from data collected from many diverse sources, even when information is extracted imperfectly, and even (in some cases) when the information is incomplete. WHIRL's ability to draw plausible inferences from incomplete data was studied systematically as an inductive classification system, and shown to be competitive with, or superior to, state-of-the-art inductive learning systems like C4.5 and RIPPER. Finally we showed that WHIRL can be used for extraction of data from HTML pages, and further, that simple heuristics written in WHIRL can be used quite effectively to aid in constructing wrappers.

The use of approximate matching for integrating heterogeneous data sources is not new [28, 13, 21, 20, 18]; however, unlike WHIRL, most previous techniques are not intended to be domain-independent. (One notable exception to this is the use of the Smith-Waterman edit distance metric [26].) WHIRL's principle innovation compared to this line of research is the adoption of term-based methods from IR for approximate matching. As well as having robust behavior for many domains, these have the advantage of efficiency; most other approximate matching schemes are relatively expensive, and perhaps for this reason, have not been used extensively in implemented Web-based information integration systems [24, 16, 1, 3, 17, 23, 25, 33]. WHIRL is also the first system of this sort to integrate approximate matching with deduction. This is a crucial step because, as shown in Section 4.3, correct inference is sometimes possible on the basis of *incorrect* matches; these inferences would be missed if matching were done off-line.

Semantically, WHIRL is closely related to earlier formalisms for probabilistic logics; in particular, if similarities were stored in an EDB predicate *sim(X,Y)* instead of being computed "on the fly", and certain irredundancy assumptions are made, then WHIRL is a strict subset of Fuhr's probabilistic Datalog [15]. There are also close connections to existing formalisms for probabilistic relational databases (*e.g.*, [2]), and earlier hybrid databases that integrate statistical IR techniques with database representations (*e.g.*, [32, 15]). The principle contribution of WHIRL is the direct use of IR-style similarities in a "soft" logic, together with the development an efficient implementation of similarity joins. This innovation allows WHIRL to be used directly for domain-independent approximate matching of keys, which greatly facilitates information integration and related tasks.

Future work on WHIRL will center on two important issues. The first is building on WHIRL's integration of extraction and deduction to build better and more flexible knowledge integration systems. The WHIRL-based systems implemented to date have all been limited by the assumption that all relevant facts have been extracted before any queries are issued; by extending WHIRL to include network access primitives, this assumption could be relaxed. The second issue is integration of WHIRL with learning methods: as an example, it would be desirable to learn to combine the heuristic ranking schemes described in Section 5.2 into a single, integrated, wrapper-generation system. Recent work on learning to combine ranking schemes [12, 14] may be relevant to this goal.

References

[1] Yigal Arens, Craig A. Knoblock, and Chun-Nan Hsu. Query processing in the SIMS information mediator. In Austin Tate, editor, *Advanced Planning Technology*. AAAI Press, Menlo Park, CA, 1996.

[2] Daniel Barbara, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Transations on knowledge and data engineering*, 4(5):487–501, October 1992.

[3] R. J. Bayardo, W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishan, A. Unruh, and D. Woelk. Infosleuth: an agent-based semantic integration of information in open and dynamic environments. In *Proceedings of the 1997 ACM SIGMOD*, May 1997.

[4] William W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Tahoe, California, 1995. Morgan Kaufmann.

[5] William W. Cohen. Learning with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996.

[6] William W. Cohen. Knowledge integration for structured information sources containing text (extended abstract). In *The SIGIR-97 Workshop on Networked Information Retrieval*, 1997.

[7] William W. Cohen. Data integration using similarity joins and a word-based information representation language. Submitted for publication., 1998.

[8] William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of ACM SIGMOD-98*, Seattle, WA, 1998.

[9] William W. Cohen. Reasoning about textual similarity in information access. Submitted for publication., 1998.

[10] William W. Cohen. A Web-based information system that reasons with structured collections of text. In *Proceedings of Autonomous Agents-98*, St. Paul, MN, 1998.

[11] William W. Cohen and Haym Hirsh. Joins that generalize: Text categorization using WHIRL. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 169–173, New York, NY, 1998.

[12] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. In *Advances in Neural Processing Systems 10*, Denver, CO, 1997. MIT Press.

[13] I. P. Felligi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 64:1183–1210, 1969.

[14] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. In *Machine Learning: Proceedings of the Fifteenth International Conference (ICML '98)*, Madison, WI, 1998. Morgan Kaufmann.

[15] Norbert Fuhr. Probabilistic Datalog—a logic for powerful retrieval methods. In *Proceedings of the 1995 ACM SIGIR conference on research in information retrieval*, pages 282–290, New York, 1995.

[16] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS approach to mediation: Data models and languages (extended abstract). In *Next Generation Information Technologies and Systems (NGITS-95)*, Naharia, Israel, November 1995.

[17] Michael Genesereth, Arthur Keller, and Oliver Dushka. Infomaster: an information integration system. In *Proceedings of the 1997 ACM SIGMOD*, May 1997.

[18] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD*, May 1995.

[19] Chun-Nan Hsu. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *Papers from the 1998 Workshop on AI and Information Integration*, Madison, WI, 1998. AAAI Press.

[20] Scott Huffman and David Steier. Heuristic joins to integrate structured heterogeneous data. In *Working notes of the AAAI spring symposium on information gathering in heterogeneous distributed environments*, Palo Alto, CA, March 1995. AAAI Press.

[21] B. Kilss and W. Alvey (ed). Record linkage techniques—1985. Statistics of Income Division, Internal Revenue Service Publication 1299-2-96. Available from http://www.bts.gov/fcsm/methodology/, 1985.

[22] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Osaka, Japan, 1997.

[23] Zoe Lacroix, Arnaud Sahuguet, and Raman Chandrasekar. User-oriented smart-cache for the web: what you seek is what you get. In *Proceedings of the 1998 ACM SIGMOD*, June 1998.

[24] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB-96)*, Bombay, India, September 1996.

[25] Giansalvatore Mecca, Paolo Atzeni, Alessandro Masci, Paolo Merialdo, and Giuseppe Sindoni. The ARANEUS web-base management system. In *Proceedings of the 1998 ACM SIGMOD*, June 1998.

[26] A. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *The proceedings of the SIGMOD 1997 workshop on data mining and knowledge discovery*, May 1997.

[27] Ion Muslea, Steven Minton, and Craig Knoblock. Wrapper induction for semistructured, web-based information sources. In *Proceedings of the Conference on Automated Learning and Discovery (CONALD)*, 1998.

[28] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, 1959.

[29] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[30] J. Ross Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann, 1994.

[31] Gerard Salton, editor. *Automatic Text Processing.* Addison Welsley, Reading, Massachusetts, 1989.

[32] Peter Schäuble. SPIDER: A multiuser information retrieval system for semistructured and dynamic data. In *Proceedings of the 1993 ACM SIGIR conference on research in information retrieval*, pages 318–327, Pittsburgh, PA, 1993.

[33] Anthony Tomasic, Remy Amouroux, Philippe Bonnet, and Olga Kapitskaia. The distributed information search component (Disco) and the World Wide Web. In *Proceedings of the 1997 ACM SIGMOD*, May 1997.

[34] Y. Yang and C.G. Chute. An example-based mapping method for text classification and retrieval. *ACM Transactions on Information Systems*, 12(3), 1994.