The WHIRL Approach to Integration: An Overview

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

Abstract

We describe a new integration system, in which information sources are converted into a highly structured collection of small fragments of text. Database-like queries to this structured collection of text fragments are approximated using a novel logic called WHIRL, which combines inference in the style of deductive databases with ranked retrieval methods from information retrieval. WHIRL allows queries that integrate information from information sources, without requiring the extraction and normalization of object identifiers that can be used as keys; instead, operations that in conventional databases require equality tests on keys are approximated using IR similarity metrics for text. This leads to a reduction in the amount of human engineering required to field an integration system.

Introduction

Knowledge integration systems like the Information Manifold (Levy, Rajaraman, & Ordille 1996), TSIMMIS (Garcia-Molina *et al.* 1995), and others (Arens, Knoblock, & Hsu 1996; Atzeni, Mecca, & Merialdo 1997) allow complex database-like queries to be posed; in particular queries that integrate information from multiple Web sites can be formulated and answered. However, current knowledge integration systems require extraction of database-like information from the Web, a relatively expensive process in terms of human engineering. The amount of human effort involved in integrating a new information source is far greater than the corresponding effort required by search engines like Lycos and Altavista; these systems allow a large portion of the Web to be queried, but the query language is limited to keyword searches on single documents.

The system described in this paper investigates an intermediate point between these two models. A set of Web information sources is converted into a highly structured collection of small fragments of text. We then approximate database-like queries to this structured collection of text fragments using a novel logic called WHIRL, which combines inference in the style of deductive databases with ranked retrieval methods from information retrieval. WHIRL allows queries that integrate information from multiple Web sites, *without* requiring the extraction and normalization of object identifiers that can be used as keys. Instead, operations that in conventional databases require equality tests on keys are approximated using IR similarity metrics for text.

In the remainder of the paper, we will first describe and motivate this text-oriented data model, and summarize the the WHIRL logic. We will then briefly review the technical results we have obtained to date using WHIRL.

An Overview of WHIRL

The underlying representation and ideas

In our system, all information is represented internally using a relational model in which every element of every tuple is assumed to contain free text. We call this data model STIR, for Simple Texts in Relations—"simple" emphasizing that the free text is assumed to have no additional structure. As an example, Figure 1 shows two STIR relations, one containing movie listings, and one containing a list of movie reviews.

The idea of incorporating free text in a relational model is, of course, not new. We carry the idea further, however, by assuming that *all* data is stored as free text. Furthermore, we do *not* assume that two text descriptions of the same object will necessarily be identical. Instead we believe that situations of the sort shown in the figure are more typical; here a movie object is described in one relation by its title, and in another by its title and year of release.

These assumptions imply that many traditional database operations cannot be performed—for instance, two relations r and q can no longer be joined by selecting tuples that have identical keys from the Cartesian product $r \times q$. However, many traditional database operations can be approximated if one assumes an accurate *similarity metric* on fragments of text. For instance, if the relations r and q both contain a field d that corresponds to a free-text object description, then the join of r and q can be approximated by sorting the tuples in $r \times q$ in decreasing order according to the similarity of $r.d$ and $q.d$. If the similarity metric is accurate, then early tuples in this ranking will correspond

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: Data represented as simple texts in relations

to "correct" pairings (*i.e.*, pairings for which $r.d$ and $q.d$ describe the same object), and the "incorrect" pairings will appear later in the ranking.

For example, using the cosine distance metric (described below) the two descriptions of "Men in Black" in Figure 1 would have high similarity, since they have many words in common; the two descriptions of "Face/Off" and "Brassed Off" will have low, but non-zero similarity, since they share the word "Off"; and the other pairings will have zero similarity. Thus of the nine possible pairings of the tuples shown in the figure, the ranking procedure described above would present a pairing of the two "Men in Black" entries first, followed by a pairing of "Face/Off" and "Brassed Off". The end user can easily find items that would be in the conventional join by examining the proposed tuples in rank order.

A remarkable fact is that robust, general methods for measuring the similarity of passages of free text do exist. In fact, virtually all modern IR systems that perform ranked retrieval rely on such metrics to order their responses. Generally, the response of such an IR system to a query is a listing of the documents that are most similar to the query, interpreting the query as a short document.

As a consequence, "soft joins" based on IR-style similarity metrics can be quite accurate in practice. Elsewhere we report experimental results with "soft joins" on actual movie listing and movie review data extracted from the Web (Cohen 1998a). We checked each proposed pairing using the hand-coded normalization and extraction procedures used in a working demonstration of the Information Manifold (Levy, Rajaraman, & Ordille 1996). We discovered that every pairing considered "correct" by the hand-coded procedures was ranked ahead of every "incorrect" pairing—the best possible result for such a ranking system. This corresponds to a non-interpolated average precision[1] of 100%.

To summarize, the STIR representation has an important advantages over a traditional database representation: it is not necessary to normalize the fields that are to be used as keys in a join, a step which usually requires manual engineering. In the example above, for instance, it is not necessary to normalize the movie names by removing years from the review relation. To our knowledge, STIR is unique among database and knowledge base systems in being able to handle this type of heterogeneity.

A more detailed description of the representation

In the remainder of this section, we will explain how the idea behind "soft joins" can be extended into a logic capable of supporting more complex queries. First, however, we will describe the STIR representation more precisely.

A *STIR database* is a set of relations $R = \{p_1, \ldots, p_n\}$. Associated with each relation p is a tuple set $TUPLES(p)$. All the tuples in $TUPLES(p)$ have the same number of components. A tuple is written $\langle \mathbf{v_1}, \ldots, \mathbf{v_a} \rangle \in TUPLES(p)$, and each tuple component $\mathbf{v_i}$ corresponds to a fragment of free text, represented as a *document vector*.

A *document vector* is a represention for text that is commonly used in the information retrieval community (Salton 1989). We will summarize this representation below, for the sake of completeness. We assume a vocabulary T of *terms*, which will be treated as atomic. In the current implementation of WHIRL, the terms are "word stems" (morphologically inspired prefixes) produced by the Porter stemming algorithm (Porter 1980). A simple text is then represented as a vector of real numbers $\mathbf{v} \in \mathcal{R}^{|T|}$, each component of which

[1]The non-interpolated average precision a standard measure of the quality of a ranking. In this case is computed by averaging, over all ranks k containing a correct pairing, the ratio r_k/k, where r_k is the number of correct pairings in the first k proposed pairings.

corresponds to a term $t \in T$. We will denote the component of $\mathbf{v}$ 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 $\mathbf{v}$. 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 (Salton 1989) and define v_t to be zero if the term t does not occur in text represented by $\mathbf{v}$, and otherwise

$$(\log(TF_{\mathbf{v},t}) + 1) \cdot \log(IDF_t)$$

In this formula, $TF_{\mathbf{v},t}$ is the number of times that term t occurs in the document represented by $\mathbf{v}$, and $IDF_t = \frac{N}{n_t}$, where N is the total number of documents in the same column as $\mathbf{v}$, and n_t is the total number of documents in this column that contain the term t.

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

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

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

Conjunctive WHIRL queries

Access to a STIR database is via an extension of Datalog[2] called WHIRL (for Word-based Heterogeneous Information Retrieval Logic). Due to space limitations, we will describe only a subset of the full logic—the language of conjunctive queries over the database.

In WHIRL, as in Datalog, a *conjunctive query* is written $B_1 \wedge \ldots \wedge B_k$ where each B_i is a *literal*. Two types of literals are allowed. The first type, *database literals*, correspond to the literals in Datalog, and are written $p(X_1, \ldots, X_a)$, where p is the name of a STIR relation, and the X_i's are variables. WHIRL also includes *similarity literals*, which are written $X \sim Y$, where X and Y are variables. Intuitively, a similarity literal can be interpreted as a requirement that documents X and Y be similar.

The semantics of WHIRL are best described in terms of substitutions.[3] The answer to a Datalog query is typically the set of ground substitutions that make the query "true" (*i.e.*, provable against the database). 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.

Given a database, we define the *score* of a ground substitution θ for a literal B as follows. If B is a database literal $p(X_1, \ldots, X_a)$, then $SCORE(B, \theta) = 1$ if $B\theta$ is a fact in the database (*i.e.*, if $\langle X_1\theta, \ldots, X_a\theta\rangle$ is in $TUPLES(p)$), and $SCORE(B, \theta) = 0$ otherwise. If B is a similarity literal $X \sim Y$, then $SCORE(B, \theta) = SIM(\mathbf{x}, \mathbf{y})$, where $\mathbf{x} = X\theta$ and $\mathbf{y} = Y\theta$. 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 conjunctive query $W = B_1 \wedge \ldots \wedge B_k$ is defined to be the product of the scores of the conjuncts:

$$SCORE(W, \theta) = \prod_{i=1}^{k} SCORE(B_i, \theta)$$

Note that all scores must be between 0 and 1.

Finally, an *answer to a WHIRL query W* is an ordered list of all substitutions θ with positive score, presented in non-increasing order by score. In other words, the answer includes all substitutions that make the database literals true, and result in a non-zero score for all the similarity literals, with the highest-scoring substitutions given first.

For example, the "soft join" of the relations in Figure 1 might be written:

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

As in the "soft join" discussed above, the result of 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. Allowing for a moment the syntactic sugar of allowing constants to appear in queries, a more complex query might search reviews by plot to see if the latest science fiction comedy is playing at the Mountainside Theater:

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

It is also possible to use WHIRL to define and materialize a *view*. We will now briefly summarize WHIRL's treatment of views, using the following example:

v(Movie)$\leftarrow$
    review(Movie,Review)
    $\wedge$ *Review$\sim$"comedy with space aliens"*

[2]Datalog refers to Prolog with no function symbols, other than constants appearing in ground literal (Ullman & Widom 1997).

[3]A *substitution* θ is a mapping from variables to document vectors. We will write a substitution as $\theta = \{X_i = \mathbf{v}_i, \ldots, X_n = \mathbf{v}_n\}$, where each X_i is mapped to the vector $\mathbf{v}_i$. The variables X_i in the substitution are said to be *bound* by θ. If W is a WHIRL query (or a literal or variable) then $W\theta$ denotes the result of applying that mapping to W—*i.e.*, the result of taking W and replacing every variable X_i appearing in W with the corresponding document vector $\mathbf{v}_i$. A substitution θ is *ground for W* if $W\theta$ contains no variables.

To materialize such a view, the interpreter will first collect the K highest-scoring answer substitutions for the conjunctive query corresponding to the body of the view clause (where K is a parameter selected by the user.) Call these K best substitutions $\Theta = \{\theta_1, \ldots, \theta_K\}$. To materialize the view above, the interpreter will create a new relation v that contains tuples of the form $\langle Movie\theta_i \rangle$, where $\theta_i \in \Theta$. Let m be some particular movie (say, m="Men in Black"). The score of the tuple $\langle m \rangle$ is computed using the formula

$$1 - \prod_{j=1}^{k_m} (1 - s_j) \quad (1)$$

where $s_1, \ldots, s_{k_m}$ are the scores of the substitutions $\theta_i \in \Theta$ such that $Movie\theta_i = m$.

One way of thinking of Θ is as a relation with one column for each variable in the body of the clause, and one row for each answer substitution for the clause body. Materializing a view can now be described as a two-step process. First, one computes this relation. Second, one "projects away" the unwanted columns (*i.e.*, the columns corresponding to variables that do not appear in the head of the clause.) The projection step can lead to duplicate rows, which are then collapsed together and rescored using Equation 1. The size of the intermediate table is limited to K rows for efficiency reasons.

WHIRL also supports disjunctive *views*. Views are discussed in more detail in (Cohen 1998a).

Experiments with WHIRL

Efficiency and Scalability

A detailed description of the inference algorithm used in WHIRL is beyond the scope of this paper (but see (Cohen 1998a) for a detailed description). The current inference algorithm exploits a number of features to make inference efficient. In particular, in most contexts it is sufficient to return a small prefix of the ordered list of answer substitutions to a user—for instance, the top 20 or so answer substitutions is usually enough. By taking advantage of special properties of the similarity function used, it is possible to compute the top-scoring k answer substitutions without computing all possible answer substitutions. This leads to a substantial speedup in answering complex queries. The current implementation of WHIRL combines A$*$ search methods with various optimization methods used in more conventional IR systems, notably the "maxscore" optimization method proposed by Turtle and Flood (Turtle & Flood 1995).

Table 1 summarizes the runtime performance[4] of WHIRL on 10 sample queries in one implemented domain. (The bird domain described in Section .) For each query, the top 20 answers were retrieved. The first column is the size of the cross-product of the relations combined by WHIRL in the query (in millions

[4]On an SGI Challenge with 250 MHz R10000 processors.

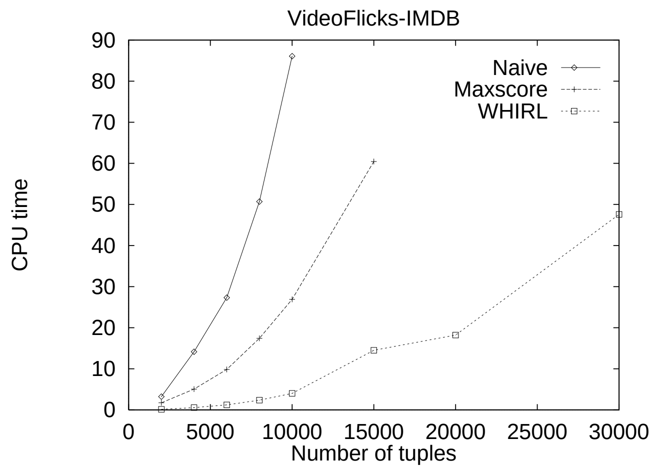

Figure 2: Runtime for similarity joins (in seconds)

of tuples). This information is provided to show that the interpreter is much more efficient than simply reordering all possible tuples from the cross-product of the relations; clearly, WHIRL is quite fast, even when computing three- and four-way joins (Cohen 1998b).

In a second set of experiments concerning scalability, we took two large relations (both containing movie names) from the Web, and evaluated the performance of WHIRL on *similarity joins*, *i.e.* queries of the form

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

We joined size n subsets of the movie relations, for various values of n between 2000 and 30,000, retrieving in each case the top 10 answers. We compared the run-time of WHIRL to the run-time for two strawmen approaches to computing similarity joins. The first, the *naive method for similarity joins*, takes each document in the i-th column of relation p in turn, and submits it as a IR ranked retrieval query to a corpus corresponding to the j-column of relation q. The top K results from each of these IR queries are then merged to find the best K pairs overall. This approach uses inverted indices, but employs no special query optimizations. The second strawman is the *maxscore method for similarity joins*; this method is analogous to the naive method described above, except that the *maxscore* optimization (Turtle & Flood 1995) is used in finding the best K results from each "primitive" query. As noted above, WHIRL is closely related this optimization method.

The results are shown in Figure 2. For this data, WHIRL speeds up the *maxscore* method by a factor of between 4 and 9, and speeds up the naive method by a factor of 20 or more. The absolute time required to compute the join is also fairly modest—with $n = 30,000$, WHIRL takes well under a minute[5] to pick the best 10 answers from the 900 million possible

[5]All timing results are given in CPU seconds on a MIPS Irix 6.3 with 200 MHz R10000 processors.

×-size (millions)	Time (sec)	Query
0.087	0.04	na(Ord,N),Ord∼"passeriforms", es(ESN,Stat),ESN∼N
0.17	0.01	na(Ord,N),Ord∼"falconiformes",fct(FN,HRef),FN∼N
16	0.02	na(Ord,N),Ord∼"gruiformes",fct(FN,HRef),FN∼N,es(ESN,Stat),ESN∼N
60	0.08	na(Ord,N),ident(IN,HRef),IN∼N,es(ESN,Stat),ESN∼N
67	0.08	na(Ord,N),N∼"strigiformes",call(CN,HRef),CN∼N,nhp(NHPN),NHPN∼N
69	0.04	na(Ord,N),N∼"anseriformes",img(IN,HRef),IN∼N,nj(NJN),NJN∼N
190	0.03	na(Ord,N),Ord∼"falconiformes",img(IN,HRef),IN∼N,es(ESN,Stat),ESN∼N
6700	0.06	na(Ord,N),bmap(BN,HRef),BN∼N,nj(NJN),NJN∼N,w(WN),WN∼N
9,000	0.08	na(Ord,N),smap(MN,HRef),MN∼N,nj(NJN),NJN∼N,w(WN),WN∼N
66,000	0.02	na(Ord,N),Ord∼"pelicaniforms",img(IN,HRef),IN∼N, nj(NJN),NJN∼N,es(ESN,Stat),ESN∼N

Table 1: Performance of the WHIRL interpreter on sample queries. Predicate names have been abbreviated for formatting reasons.

candidates. For more detail on these experiments, see (Cohen 1998a).

Accuracy of Integration Inferences

We also evaluated the accuracy of similarity joins, again using data taken from the Web. We selected pairs of relations which contained two or more plausible "key" fields. 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. 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. The results for three such pairs of relations are summarized in Table 2. On these domains, similarity joins are extremely accurate, relative to the secondary keys (Cohen 1998a).

In another set of experiments (Cohen 1997), we looked at *constrained similarity joins*—queries of the form

$$
\begin{aligned}
&p(X_1,\ldots,X_i,\ldots,X_a)\\
&\wedge\ q(Y_1,\ldots,Y_j,\ldots,Y_b)\\
&\wedge\ X_i \sim Y_j\\
&\wedge\ X_k \sim \text{“constraint”}
\end{aligned}
$$

In this case the similarity join is restricted to tuples for which some field is similar to some "constraint document" specified by the user. This is analogous to imposing an additional selection criterion on an ordinary relational join. These queries are somewhat difficult to evaluate automatically in general, so we were careful about choosing the additional "selectional criteria"; in the movie domain, for instance, the queries specified that the review must be similar to a plot description which was intended to pick out one particular movie. (For more details see (Cohen 1997).) The average precision for these queries is lower, but still respectable—between 36% and 100%, and averaging around 56%. We note that these queries involve combining a similarity join with an IR-type ranked retrieval, so performance better than that achievable by modern IR systems on ranked retrieval problems is impossible.

Integration of Partially Extracted Data

A major advantage of the STIR representation is that it is very robust to errors in the data. To illustrate this, consider the problem of working with fields that have been incompletely extracted.

In the movie domain experiment described in Section (and reported again above) we used the same hand-coded extraction procedures used in the Information Manifold. Thus each movie listing was segmented as shown in Figure 1, with the movie title in a separate field from other information; similarly, in the review relation, the name of the movie being reviewed was also extracted. However, similarity joins can also be performed when object descriptions *cannot* be easily extracted from the surrounding text. For example, one could treat the entire review as a single field; similarly, one could treat an entire movie listing as a single unsegmented field including movie name, cinema name, and show times. Similarity joins can still be performed on the resulting fields, and documents referring to the same movie should still match reasonably well, although one would expect the irrelevant "noise" words to have some adverse affect.

In experiments with the review and movie listing relations, the degradation due to incomplete extraction was quite small. Joining movie names to unsegmented movie reviews reduces non-interpolated average precision by only 2%, to 98%, and joining unsegmented movie listings to unsegmented reviews reduces non-interpolated average precision by less than 7%, to 93%.

These intriguing results have now been duplicated in a second domain. We took a Web page that listed online demos of educational games for children, and at-

Similarity Joins

Domain	Average Precision
Business	84.6%
Animals	92.1%
Movies	100.0%

Constrained Similarity Joins

Constraint	Domain	Average Precision
modems	business	71.5%
internet	business	100.0%
software	business	57.0%
telecom. equip.	business	49.0%
plot1	movies	54.5%
plot2	movies	61.3%
mexico	animals	48.3%
texas	animals	46.5%
washington	animals	36.2%
florida	animals	42.2%
maine	animals	50.0%

Table 2: Average precision for similarity joins

tempted to join it with second list of education games. The second list was from a relation providing information concerning the age range for which a game is considered appropriate, and will be called henceforth the *agelist*. The original demo listing was simply an itemized list of free-text descriptions. Some representative items, with hyperlinks removed, are given below:

- *7th Level has The Great Word Adventure Demo starring Howie Mandel for Windows.*
- *Conexus has two shockwave demos - Bubbleoids (from Super Radio Addition with Mike and Spike) and Hopper (from Phonics Adventure with Sing Along Sam).*
- *Broderbund software has the Zoombinis Mudball Wall Demo. (Mac and Win) Cool!*

We performed two experiments with this data. First, we joined each complete list item with the *agelist*, and recorded a pairing as correct if the *agelist* game was in fact mentioned in the list item. Second, we manually extracted each game name from the page, and joined the resulting list of game names with *agelist*. A pairing was judged correct if the game names referred to the same game (as determined by manual inspection.)

In this case the average precision was 86.5% with extraction, and 67.1% without extraction. Although extraction does provide a benefit in this case, it should be noted that even without extraction, the result of the similarity join is certainly accurate enough to be usable. For instance, without extraction, the first 23 pairings contain only two mistakes; these mistakes, appearing at ranks 5 and 6, incorrectly pair the games "Mario Teaches Typing" and "Mario Teaches Typing 2".

Accuracy of Classification Inferences

We also evaluated the accuracy of a second type of query. Suppose one is given a relation 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. (For instance, the instances in *train* might be news story headlines, and the labels might be subject categories from the set *business,sports,US,world,other.*) One can assign a label to a new instance (*e.g.*, a new headline) by first making it the sole entry in the relation *test(inst)*, and then materializing the view

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

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 similar to the vector space distance-weighted K-NN method investigated by Yang (Yang & Chute 1994); however, WHIRL uses a different scheme is used to combine the weights associated with the K closest neighbours of an unclassified instance.

As a further evaluation of WHIRL, we created nine benchmark problems from web pages found on the World Wide Web; all of these associate text, typically 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's method, C4.5 (Quinlan 1994) using a binary representation, and Ripper (Cohen 1995; Cohen & Singer 1996) using set-valued features. We used $K = 30$ for WHIRL and Yang'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 was used to assess the accuracy of the predictions; for details see (Cohen & Hirsh 1998).

The results are shown in Table 3. WHIRL outperforms C4.5 eight of nine times, and outperforms RIPPER seven of nine times. The competing method that is closest in performance to WHIRL is Yang's method, which is closely related to WHIRL. WHIRL also outperforms Yang's method eight of nine times; although all of these eight differences are small, five of them are statistically significant. WHIRL is never significantly worse

Accuracy (%)					
	default	RIPPER	C4.5	Yang	WHIRL
memos	19.8	50.9	57.5	64.4	66.5
cdrom	26.4	38.3	39.2	47.2	47.5
birdcom	42.8	88.8	79.6	82.3	83.9
birdsci	42.8	91.0	83.3	89.2	90.3
hcoarse	11.3	28.0	30.2	32.7	32.9
hfine	4.4	16.5	17.2	21.0	20.8
species	51.8	90.6	89.4	93.6	94.3
books	5.7	42.3	52.2	60.1	61.4
netvet	22.4	67.1	68.8	67.8	68.1
average	25.24	57.52	57.41	62.00	62.90

Time (sec)			
	RIPPER	C4.5	WHIRL
hcoarse	92.7	51.4	21.1
hfine	126.0	55.7	22.0
species	299.7	1505.6	23.5
books	213.4	11437.2	72.4
netvet	122.2	1566.1	37.8
average	170.8	2923.2	35.4

Table 3: Accuracy, and runtime for learning and classification

than any other learner on any individual problem.[6]

We also observe that although nearest neighbour methods are often slow in terms of classification time, the indexing methods employed by WHIRL (and other IR systems) for text make nearest-neighbour classification quite efficient for problems of this type; this is indicated by Table 3, which also gives the combined time required to learn from the training data and classify the test data for the five larger benchmark problems.[7]

Practical experience in using WHIRL

WHIRL has also been used as the central component of a working integration system. The main additional components of this system are an HTTP server interface to WHIRL, which allows conjunctive queries to WHIRL to be easily formulated using HTML forms; and a spider program which allows one to easily extract STIR relations from HTML pages. These components are described in detail elsewhere (Cohen 1998b); here we will simply observe that the ability to work effectively with incompletely extracted data makes extraction from HTML pages much easier.

At the time of this writing, two moderately large domains have been fully implemented for this system, and a number of isolated information sources have been converted for purposes of evaluation or demonstration. One implemented domain integrates information on birds of North America from about two dozen sites. The integrated database contains pointers to over 5000 pages containing information about birds, including nearly 2800 images of birds; around 700 fact sheets; more than 350 bird calls; over 1000 maps showing relative abundance of bird species at various times of the year; a list of bird species that are endangered in the United States; and bird checklists for North America, the state of New Jersey, and one national park. The second domain integrates information about educational computer games from twelve sites. The integrated database contains more than 2500 pointers to various web pages, including over 1800 reviews, pricing information for over 300 games, pointers to more than 60 online demos, and over 280 home pages for game publishers. Both domains are available as demos on the World Wide Web at http://whirl.research.att.com.

Conclusions

WHIRL differs from previous knowledge integration systems in using an unconventional data model which represents information directly in text, and answering queries using a "soft" logic, WHIRL. WHIRL approximates conventional database queries using textual similarity tests. In particular, information sources are represented as tables of simple texts, and joins and other database queries are approximated by replacing equality tests with similarity tests on text. WHIRL uses the cosine distance metric and TF-IDF weighting schemes to estimate the similarity of texts; this is one of several robust and general similarity metrics developed by the IR community.

Representing data as tables of texts makes knowledge integration easier in several ways. Notably, the fields that will be used as keys do not need to normalized. Furthermore, the robustness of the similarity metrics means that extraction need not be complete—acceptable performance is still obtained if some "noise" words are contained in a field. These properties make it much easier to convert information sources into our data model.

WHIRL has been validated with a number of experiments in controlled settings, many of which are described above. These controlled settings are in many respects artificial: we do not expect, for instance, that queries in domains of real interest will often be simple similarity joins, nor do we believe that WHIRL will be typically used as a nearest-neighbour classifier. However, it is difficult to quantify the performance of

[6] We used McNemar's test to test for significant differences on the problems for which a single holdout was used, and a paired t-test on the folds of the cross-validation when 10CV was used.

[7] As a brief indication of size, *hcoarse* has 1875 training examples and 600 test examples, 126 classes, and 2098 distinct terms; *books* has 3501 training examples, 1800 test examples, 63 classes, and 7019 terms.

WHIRL's similarity-based reasoning methods on complex queries, and WHIRL's excellent performance on these simple types of queries is extremely encouraging. WHIRL has also been validated by building working information access systems in two different non-trivial domains.

References

Arens, Y.; Knoblock, C. A.; and Hsu, C.-N. 1996. Query processing in the SIMS information mediator. In Tate, A., ed., *Advanced Planning Technology.* Menlo Park, CA: AAAI Press.

Atzeni, P.; Mecca, G.; and Merialdo, P. 1997. Semistructured and structured data on the Web: going back and forth. In Suciu, D., ed., *Proceedings of the Workshop on Management of Semistructured Data.* Tucson, Arizona: Available on-line from http://www.research.att.com/ suciu/workshop-papers.html.

Cohen, W. W., and Hirsh, H. 1998. Joins that generalize: Text categorization using WHIRL. Submitted to KDD-98.

Cohen, W. W., and Singer, Y. 1996. Context-sensitive learning methods for text categorization. In *Proceedings of the 19th Annual International ACM Conference on Research and Development in Information Retrieval*, 307–315. Zurich, Switzerland: ACM Press.

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

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

Cohen, W. W. 1998a. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of ACM SIGMOD-98.*

Cohen, W. W. 1998b. A Web-based information system that reasons with structured collections of text. In *Proceedings of Autonomous Agents-98.*

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

Levy, A. Y.; Rajaraman, A.; and Ordille, J. J. 1996. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB-96).*

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

Quinlan, J. R. 1994. *C4.5: programs for machine learning.* Morgan Kaufmann.

Salton, G., ed. 1989. *Automatic Text Processing.* Reading, Massachusetts: Addison Welsley.

Turtle, H., and Flood, J. 1995. Query evaluation: strategies and optimizations. *Information processing and management* 31(6):831–850.

Ullman, J., and Widom, J. 1997. *A first course in database systems.* Upper Saddle River, New Jersey: Prentice Hall.

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