A Web-based Information System that Reasons with Structured Collections of Text

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

Abstract

The degree to which information sources are pre-processed by Web-based information systems varies greatly. In search engines like Altavista, little pre-processing is done, while in "knowledge integration" systems, complex site-specific "wrappers" are used integrate different information sources into a common database representation. In this paper we describe an intermediate between these two models. In our system, 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 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. This leads to a reduction in the amount of human engineering required to field a knowledge integration system. Experimental evidence is given showing that many information sources can be easily modeled with WHIRL, and that inferences in the logic are both accurate and efficient.

1 Introduction

There are (at least) two different models for systems designed to access information on the Web. The first model, which might be called the information retrieval (IR) model, is exemplified by search engines like Lycos and Altavista. In this model a large portion of the Web can be queried, but the query language is limited to keyword searches on single documents. The second model is exemplified by systems like the Information Manifold [14], TSIMMIS [9], and others [2, 4]. This "knowledge integration model" supports more complex database-like queries; in particular queries that integrate information from multiple Web sites can be formulated. However, current knowledge integration systems require extraction of database-like information from the Web, a relatively expensive process in terms of human engineering.

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 relevant properties of the WHIRL logic. We will then describe how WHIRL is embedded in an Web-based information system, and present data on the cost of integrating new information sources and the speed of inference.

2 An Overview of WHIRL

2.1 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

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

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 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.[1] As a consequence these "soft joins" 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 [6]. We checked each proposed pairing using the hand-coded normalization and extraction procedures used in a working demonstration of the Information Manifold [14]. 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[2] of 100%.

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

In the experiment described above, we used the same hand-coded extraction procedures used in the Information Manifold. Thus each movie listing was segmented as shown in the figure, 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%. Similar levels of average precision were achieved in joining other types of objects [6].

To summarize, the STIR representation offers several advantages over a traditional database representation. One advantage is that extraction procedures can be approximate or incomplete—as is illustrated by the experiments reported above. A second advantage is that 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.

2.2 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*.

[1] 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.

[2] 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.

A *document vector* is a represention for text that is commonly used in the information retrieval community [17]. 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 [16]. A simple text is then represented as a vector of real numbers $\mathbf{v} \in \mathcal{R}^{|T|}$, each component of which 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 [17] and define v_t to be zero if the term t does not occur in text represented by $\mathbf{v}$, and otherwise

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

In this formula, $TF_{\mathrm{v},t}$ 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

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

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 since all document vectors are of unit length, and have non-negative components, $SIM(\mathbf{v}, \mathbf{w})$ is always between zero and one.

2.3 Conjunctive WHIRL queries

Access to a STIR database is via an extension of Datalog[3] 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.[4] 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"*

2.4 Implementation considerations

While a detailed description of the inference algorithm used in WHIRL is beyond the scope of this paper, a few points are worth making. In most contexts it is reasonable for a theorem-prover to return a small prefix of this ordered list—for instance, the top 20 or so answer substitutions. 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. The inference algorithm WHIRL uses generalizes the "maxscore"

[3] Datalog refers to Prolog with no function symbols, other than constants appearing in ground literal [21].

[4] 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.

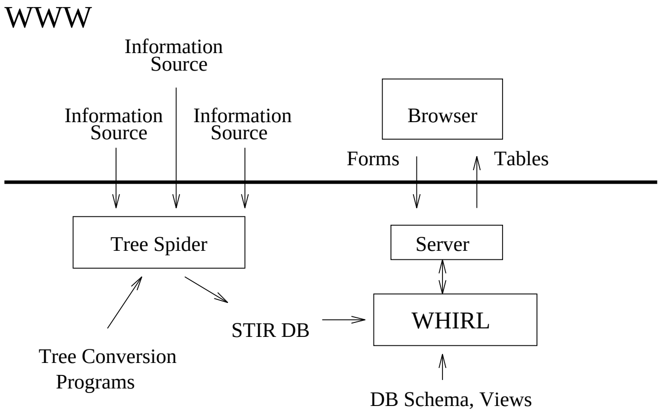

Figure 2: Architecture of the Information Access System

optimization method proposed by Turtle and Flood [20], and is described in detail elsewhere [6].

The current implementation of WHIRL also supports a somewhat more expressive query language than that described above; for instance, a query can be a disjunction of several conjunctive queries. WHIRL also supports *views*, which can be used to provide a more uniform interface to heterogeneous data sources. In the applications we have considered to date, however, only part of this extra functionality has been needed.

3 A Web Information System Based on WHIRL

We will now describe how WHIRL has been embedded in a Web-based information access system. In particular, we will discuss how information is collected from the Web, how WHIRL queries are formulated by the user, and how answers to queries are presented to the user.

The architecture of the overall information system, which we call SPIRAL (for Server Program for Integration and Retrieval via Logic), is shown in Figure 2. SPIRAL departs from most knowledge integration systems in collecting data off-line, rather than at query time. A spider program downloads Web pages and converts them, at download time, into STIR format; for reasons that will become clear shortly, this program is called a *tree spider* in the figure. The original HTML is then discarded, leaving a reduced database that contains document vector representations of every simple text in the database, as well as a human-readable *summary* (currently, the first 100 characters) of every simple text. A *reference* for every simple text is also recorded for later use; the reference is the URL from which the text was obtained. The information retained by the spider is thus quite similar to the information stored by search engines like Altavista.

The STIR database is then loaded into WHIRL. Access to the data is provided by a server. The server accepts an HTTP request which encodes a conjunctive WHIRL query. This encoding is designed so that HTML forms can be conveniently used to generate queries. The server then executes the query with WHIRL, and returns an HTML table containing the k top-scoring answer substitutions for the query. Each field of this table corresponds to a simple text, and as in conventional search engines, this text is represented with a summary and a reference. Links associated with the table allow the user to request the next k substitutions.

This approach could be easily extended in a number of ways. For instance, it is straightforward to use non-HTML information sources, such as flat files, by using a separate mechanism to convert these sources to STIR. (In fact, we have imported several non-HTML sources into STIR for experimental purposes.) One could also implement a system that collects information at query time and converts it to STIR; however, creating a single central database greatly simplifies query planning.

3.1 The tree spider: collecting data from the Web

The spider that is used to download pages from the Web also converts these pages to STIR. Conversion to STIR corresponds to the "wrapping" of information sources that other knowledge integration systems perform. In this section we will describe the conversion language. First, however, we will summarize the details of the target language—the data format used for STIR.

The data format we use for STIR is conventional, with one convenient extension. A tuple of the relation p is represented with a special "begin tuple" character sequence coupled with the name for p; some free text, possibly separated by a second special field-separator character sequence; and finally, an "end tuple" character sequence. As an example (using for readability LaTeX symbols as delimiters), the first tuple in Figure 1 might be written as

$\ll$ *movieListing Roberts Theaters Chatham* $\|$
Brassed Off $\|$ *7:15pm - 9:10* $\gg$

The convenient extension is that these tuples may be nested. If a tuple is nested inside a simple text $\mathbf{v}$, then that tuple will be read recursively; however, to preserve information about the nested tuple's original position, WHIRL automatically inserts into the nested tuple an additional field that is filled with a *copy* of the containing document, $\mathbf{v}$. As an example of such nested data, consider the following *venue* tuple, which contains a cinema name and show times, and has a *movieName* tuple embedded within the cinema name:

$\ll$*venue Roberts Theaters Chatham*
$\ll$*movieName Brassed Off*$\gg$ $\|$ *7:15pm - 9:10* $\gg$

This would be read in as two tuples, with the inner *movieName* tuple containing a copy of the document in which it is nested:

$\ll$*venue Roberts Theaters Chatham* $\|$ *7:15pm - 9:10* $\gg$
$\ll$*movieName Roberts Theaters Chatham* $\|$ *Brassed Off* $\gg$

WHIRL has also been extended with an "eq" predicate, which is defined so that "eq($\mathbf{v}$,$\mathbf{v}'$)" has score 1 only when $\mathbf{v}$ and $\mathbf{v}'$ are copies, and has score 0 otherwise.[5] This means that data written using nested tuples can be easily "flattened" into a tabular representation with a WHIRL view that uses the "eq" predicate. For instance, the following view could be used to generate a single table containing a movie name, cinema name, and show time:

$$\begin{aligned}&movieListing(Movie, Cinema, Times) \leftarrow\\ &\quad venue(Cinema, Times),\\ &\quad movieName(CinCopy, Movie),\\ &\quad eq(Cinema, CinCopy).\end{aligned}$$

[5] In particular, "eq($\mathbf{v}$,$\mathbf{v}'$)" has score 0 when $\mathbf{v}$ and $\mathbf{v}'$ are identical in the vector space model, but were not created as copies by the WHIRL system.

3.1.1 Simple conversion programs

The convenience of this notation in translating from HTML to STIR should be obvious. To a first approximation, an HTML document can be represented as an *HTML parse tree*—a tree in which the internal nodes are labeled with the names of HTML constituents, and the leaves are strings of text. With the nested tuple notation, STIR data can be represented with trees that are formally identical: in STIR data, the internal nodes of the tree are labeled with relation names, and again the leaves are text. Conversion of HTML to STIR is thus a process of selecting an appropriate subset of the HTML parse tree, and appropriately relabeling the internal nodes.

Adopting this model of data conversion leads to a very simple and succinct conversion language. We will first describe the basic case for this language, and then describe some extensions which are useful for more complex conversions.

In the simplest case, a conversion program is a string of the form a_1 *as* b_1—a_2 *as* b_2—$\cdots$—a_n *as* b_n, where each a_i is the name of an HTML constituent and each b_i is either a STIR relation name, or $NULL$. Each "a_i *as* b_i" construct is called a *conversion step*. (A conversion step of the form a_i *as* $NULL$ can also be written more compactly as a_i.) To convert a tree using the program a_1 *as* b_1—a_2 *as* b_2—$\cdots$—a_n *as* b_n, the following rules are followed.

- Let ν be a node in the syntax tree, and consider the sequence $\sigma(\nu)$ of node labels on the path from the root of the tree to ν. If $\sigma(\nu) = a_1 \ldots a_k$ for $k \leq n$ (that is, if this path is a prefix of the sequence of a_i's) then ν is relabeled with b_k.
- If $a_1 \ldots a_n$ is a proper prefix of $\sigma(\nu)$ (that is, if ν falls below the tree node corresponding to a_n, the final step in the program) then ν is relabeled with $NULL$.
- If neither of the cases above hold, then the subtree rooted at ν is deleted.

Finally, internal nodes labeled $NULL$ are removed from the tree.

As an example, if an HTML document containing movie listings was formatted as an unordered list (<ul>) with movie names in boldface (<b>), then the conversion program

html—body—ul—li as venue—b as movieListing

would convert the HTML into STIR data like the following:

$\ll$*venue Roberts Theaters Chatham*
$\ll$*movieListing Brassed Off*$\gg$ *7:15pm - 9:10*$\gg$

This will be read in as two tuples, with the inner *movieListing* tuple containing a copy of the *venue* document in which it is nested:

$\ll$*venue Roberts Theaters Chatham 7:15pm - 9:10* $\gg$
$\ll$*movieListing Roberts Theaters Chatham 7:15pm - 9:10* $\|$ *Brassed Off*$\gg$

This example is imaginary, but typical. Often it is the case that some but not all of the desired extraction can be performed using only HTML features—here the show times and theater names cannot be easily teased apart. Fortunately, the robustness of WHIRL makes partial extraction sufficient in many cases.

In writing conversion programs, we have found some simple programming tools to be useful. One tool takes an HTML document and outputs all paths that occur at least k times in the document, in decreasing order of frequency. If the document contains a single large regular structure (such as a large table or list) then some frequent path will correspond to the leave items in this structure. (For instance, in the example above, the paths *html—body—ul—li* and *html—body—ul—li—b* will be frequent.) In this case, writing a conversion program is extremely straightforward—it is only necessary to provide the names of the STIR predicates to which each path element should be mapped.

3.1.2 Additional constructs

Simple conversion programs provide enough power to convert a large fraction of the information sources that we have considered; however, some additional constructs have been added to aid in converting more complex pages.

HTML attributes. The tree model introduced above ignores HTML attributes, which are sometimes important; for example, it is often desirable to associate a simple text with an URL that appears as an "href" attribute of some anchor constitutent. To handle HTML attributes, the syntax for a single step in a conversion program is extended to

$$a_i(c_1, \ldots, c_k) \text{ as } b_i(c_1, \ldots, c_k)$$

Here the c_j's are the names of HTML attributes, and the conversion step specifies that their values will be copied into the first k fields of the STIR relation b_i.

We have also introduced a small number of "pseudo-attributes", which are computed by the tree spider. For instance, the pseudo-attribute "hi" for a constituent ν corresponds to the text of the most recently preceding level-i header. For example, if the name of a theater appears as a level 3 header, followed by an unordered list of movies playing at the theater, one might use the following conversion program: *html—body—ul—li(h3) as movieListing(h3).*

Escape to the host language. An optional "where E" construct can be attached to any conversion step, where E is an arbitrary executable test in the host language (Perl5). The test E is additional filter on whether a conversion step can be applied to a node; not only must the HTML tag be the same as a_i, but the test E must also succeed. Attributes and pseudo-attributes in E are replaced with appropriate values. For example, to save in the relation p only those list items that fall under a header containing the string *new*, one might use Perl's regular expression matching operator =~ in this conversion program: *html—body—ul—li(h3) as movieListing where h3=~/new/.*

Multiple paths. Allowing only one conversion path is clearly a restriction. While usually one path is sufficient, we allow more than one conversion program to be applied to the same HTML document, thus supporting documents with less regular structure. Caching on the part of the spider makes this relatively inexpensive.[6]

Non-path commands. Another problem with HTML is that, while constructs for lists and tables exist, these constructs are frequently ignored; instead authors frequently segment a page using explicit line breaks (the "
" construct), explicit horizontal rules (the "<hr>" construct), and so on. To handle such pages, we have also included a small number of constructs into the conversion language that perform more powerful tree-restructuring operations.

[6] Currently, recently downloaded documents and the HTML parse tree for the last document accessed are cached.

- A step *vsplit* a_i *as* b_i *where* E partitions the subtrees of the current node, using subtrees that satisfy the test "a_i *where* E" as dividers. Each partition is converted separately using subsequent steps in the conversion program, and each of these subtrees is placed in a different tuple of the b_i relation. As an example, if movie listings were separated only by line breaks, the HTML page might be converted into STIR by the program *html—body—vsplit br as venue–b as movieListing.*

- A step *hsplit* a_i *where* E partitions the subtrees of the current node in the same way; however, each partition is placed in a different field of the current tuple. This is mostly used for converting an HTML table into a STIR table. For instance, if the movie listings were presented as an HTML table, they could be converted by the program *html—body—table—tr as movieListing—hsplit td.*

- A step *before* a_i *where* E discards all subtrees after the first subtree with a root satisfying the test "a_i *where* E", similarly a step *after* a_i *where* E discards all subtrees before the first subtree with a root satisfying the test "a_i *where* E".

3.2 Evaluation of the tree spider language

To evaluate the expressive power of this language, we selected 38 URLs, from 23 distinct Web sites, which were used to demonstrate an early version of our system. These information sources had previously been converted to STIR using hand-coded Perl programs that traversed the HTML syntax tree. Most of these sources were index pages that summarized items of one particular type on a larger site (for instance, a list of images found on that site) and had a relatively regular format.

Four of these URLs could no longer be accessed. One could not be easily converted using the language described above—this information source made heavy use of preformatted tables, the structure of which is not reflected in a syntax tree.[7] One URL was converted by applying four different programs in succession. The remaining URLs were converted with a single one-path conversion program. (These four programs extracted list items at different levels in a nested-list structure.) The average length of the programs was 47 characters (not counting the steps *html—body—* which precede every program) and the average time to develop a program was between three and four minutes.[8]

As expected, relatively little use was made of more advanced features of the language. The 33 programs contain a total of four "where" conditions, three of which tested the type of an URL. There was one "hsplit" conversion step, one "vsplit" step, one "before" step, and no "after" steps.

Clearly, one should use care interpreting these numbers, since the sample is one of convenience. However, the experiment does show that there are many sites that can be easily modeled with this language.

3.3 Runtime performance

At the time of this writing, two domains for SPIRAL have been fully implemented, and a number of isolated information sources have been converted for evaluation purposes. 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.

Table 1 summarizes the runtime performance[9] of the SPIRAL server on 10 sample queries in the bird domain. 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 of tuples). This information is provided to show that the interpreter is much more efficient than simply re-ordering all possible tuples from the cross-product of the relations; clearly, WHIRL is quite fast, even when computing three- and four-way joins.

These queries were generated by HTML forms, which allow the user to state questions in certain restricted formats. Some of the queries have been "translated" by showing the English text appearing on the form after the user's selections have been made.

Controlled experiments also have been performed with WHIRL on other domains [6].

4 Related work

Integration of heterogeneous data sites via approximate text matching has been previously proposed by Huffman and Steier [11]. The system they propose, however, is specialized to matching proper names, and the matching procedure is not embedded in a general logic.

WHIRL is closely related to probabilistic databases (*e.g.*, [8, 5]). To our knowledge such database systems have not been used in knowledge integration tasks. There are also many technical differences between WHIRL and other "soft" database systems. The main difference is that WHIRL is based on directly on similarity, rather than probability. Similarity metrics much like the cosine difference measure can be expressed in terms of probabilities, so a probabilistic logic could be used to implement a WHIRL-like language; however, the representation of data would be less direct, and the probabilistic queries needed to emulate WHIRL queries would be more complex. To our knowledge, the WHIRL interpreter is also unique in generating only a few "best" answers to a query. Probabilistic database systems typically generate all tuples with non-zero probability; this would often be an impractically large set for problems of the sort considered here.

We have argued that the STIR data model makes it easier to integrate information sources, since normalization of fields is avoided, and extraction can be only approximate. An alternative approach to lowering the cost of "wrapping" data is to provide special languages for wrappers [10], or to partially automate wrapper generation [3, 13]. These make extraction easier for relatively regular sources, but do not completely eliminate the cost of integrating a new source:

[7] Constructs to support preformatted tables have since been added to the language.

[8] This includes the time required to download the pages, but ignores the time required to locate them initially.

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

×-size (millions)	Time (sec)	Query
0.087	0.04	na(Ord,N),Ord∼"passeriforms", es(ESN,Stat),ESN∼N "tell me the names of birds in the order passeriforms that have been sited in North America and are endangered or threatened"
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 "show me the summer range of birds that have been sited in New Jersey and are the Audubon Society's watchlist"
66,000	0.02	na(Ord,N),Ord∼"pelicaniforms",img(IN,HRef),IN∼N, nj(NJN),NJN∼N,es(ESN,Stat),ESN∼N "show me pictures of birds in the order pelicaniforms that have been sited in New Jersey and are endangered or threatened"

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

for instance, it is still necessary to normalize keys. Furthermore, even the most automatic wrapper generation tools require some user input, in the form of labeling examples, verifying hypotheses about the structure of the input, or (in the case of Kushmerick *et al.*) providing "recognizers" for various data types. We view this work is complementary to ours—ideally such automated methods would be combined with the simpler data model provided by STIR to further reduce the cost of integrating a new source.

A number of knowledge integration systems, and other systems which provide a data model for the Web, have proposed "semi-structured" data model, in which data is represented in graphs, rather than relations [18, 1, 19]. Although we have so far focused on the simpler relational models, we view work on graph-like models as complementary to the contributions of this paper. The problems resolved by graph-like data models are largely disjoint from the problems addressed by the STIR data model; in particular, adopting a graph-like data model still requires one to construct key fields via extraction and normalization.

Some languages have also been proposed that provide a database-like view of the Web (*e.g.*, [7, 15, 12]). In these database languages the base relations relate to how pages are linked together, and which words are contained on a page; by issuing database queries, certain types of spider-like operations can be easily expressed. In effect, these languages offer a means declaratively navigating the Web. Our work has instead concentrated on integrating sites that contain no explicit links connecting them, by exploiting inter-site similarities—in particular, the similarity of simple texts extracted from the sites.

5 Conclusions

The goal of knowledge integration systems is to reason with information in heterogeneous data sources, such as data sources on the Web [14, 2, 9, 4]. Unfortunately, the "lowest common denominator" representation for information found on the Web is text, which is often messy and ambiguous. Therefore knowledge integration systems which use a conventional data model must perform extensive pre-processing of the data, which is expensive in terms of human time; further, the systems are limited to relatively well-structured data.

The system described here 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: currently, we extract data from Web sites using a very simple declarative language based on applying tree-rewriting rules to parsed HTML pages.

Our approach has been validated by building working information access systems in two different domains. Together these provide access to thousands of web pages, using information collected from more than three dozen sites.

Acknowledgements

I would like to thank Edith Cohen, Haym Hirsh, Alon Levy, Amit Singhal and many other colleagues for numerous helpful discussions, and Haym Hirsh and two anonymous reviewers for helpful comments concerning the presentation of this paper.

References

[1] Serge Abiteboul and Victor Vianu. Regular path

queries with constraints. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-97)*, Tucson, AZ, May 1997.

[2] 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.

[3] Naveen Ashish and Craig Knoblock. Wrapper generation for semistructured Internet sources. In Dan Suciu, editor, *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997. Available on-line from http://www.research.att.com/~suciu/workshop-papers.html.

[4] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. Semistructured and structured data on the Web: going back and forth. In Dan Suciu, editor, *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997. Available on-line from http://www.research.att.com/~suciu/workshop-papers.html.

[5] 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.

[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] Thorsten Fiebig, Jurgen Weiss, and Guido Moerkotte. RAW: a relational algebra for the Web. In Dan Suciu, editor, *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997. Available on-line from http://www.research.att.com/~suciu/workshop-papers.html.

[8] 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.

[9] 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.

[10] J. Hammer, H. Garcia-Molina, J. Cho, and A. Crespo. Extracting semistructured information from the Web. In Dan Suciu, editor, *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997. Available on-line from http://www.research.att.com/~suciu/workshop-papers.html.

[11] 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.

[12] D. Konopnicki and O. Schmueli. W3QS: a query system for the world wide web. In *Proceedings of the 21nd International Conference on Very Large Databases (VLDB-96)*, Zurich, Switzerland, 1995.

[13] 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.

[14] 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.

[15] Alberto Mendelzon and Tovo Milo. Formal models of Web queries. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-97)*, Tucson, AZ, May 1997.

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

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

[18] Dan Suciu. Query decomposition and view maintanance for query languages for unstructured data. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB-96)*, Bombay, India, 1996.

[19] Dan Suciu, editor. *Proceedings of the Workshop on Management of Semistructured Data*. Available on-line from http://www.research.att.com/~suciu/workshop-papers.html, Tucson, Arizona, May 1997.

[20] Howard Turtle and James Flood. Query evaluation: strategies and optimizations. *Information processing and management*, 31(6):831–850, November 1995.

[21] Jeffrey Ullman and Jennifer Widom. *A first course in database systems*. Prentice Hall, Upper Saddle River, New Jersey, 1997.