Data Integration Using Similarity Joins and a Word-Based Information Representation Language

WILLIAM W. COHEN
AT&T Labs—Research, Shannon Laboratory

The integration of distributed, heterogeneous databases, such as those available on the World Wide Web, poses many problems. Here we consider the problem of integrating data from sources that lack common object identifiers. A solution to this problem is proposed for databases that contain informal, natural-language "names" for objects; most Web-based databases satisfy this requirement, since they usually present their information to the end-user through a veneer of text. We describe WHIRL, a "soft" database management system which supports "similarity joins," based on certain robust, general-purpose similarity metrics for text. This enables fragments of text (*e.g.*, informal names of objects) to be used as keys. WHIRL includes textual objects as a built-in type, similarity reasoning as a built-in predicate, and answers every query with a list of answer substitutions that are ranked according to an overall score. Experiments show that WHIRL is much faster than naive inference methods, even for short queries, and efficient on typical queries to real-world databases with tens of thousands of tuples. Inferences made by WHIRL are also surprisingly accurate, equaling the accuracy of hand-coded normalization routines on one benchmark problem, and outperforming exact matching with a plausible global domain on a second.

Categories and Subject Descriptors: H.2.5 [**Information Systems**]: Database Management—*heterogeneous databases*; H.2.3 [**Information Systems**]: Database Management—*data manipulation languages*; *query languages*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*retrieval models*; *performance evaluation*

General Terms: Reliability

1. INTRODUCTION

Integration of distributed, heterogeneous databases, sometimes known as *data integration*, is an active area of research in the database community [Duschka and Genesereth 1997b; Levy et al. 1996b; Arens et al. 1996; Garcia-Molina et al. 1995; Tomasic et al. 1997; Bayardo et al. 1997]. Largely inspired by the proliferation of database-like sources on the World Wide Web, previous researchers have addressed a diverse set of problems,

Author's address: WhizBang Labs, 4616 Henry Street, Pittsburgh, PA 15213; email: wcohen@whizbang.com.

ranging from access to "semi-structured" information sources [Suciu 1996; Abiteboul and Vianu 1997; Suciu 1997] to combining databases with differing schemata [Levy et al. 1996a; Duschka and Genesereth 1997a]. Data integration is analogous to the problem of *collection fusion*—integration of distributed text collections—but differs in that the information sources to be integrated are structured.

In this paper we will consider a new type of data integration problem, namely, the problem of combining information from relations that lack common formal object identifiers. To illustrate this problem, consider a relation p with schema p(company,industry) that associates companies with a short description of their industries, and a second relation q with schema q(company,website) that associates companies with their home pages. If p and q are taken from different, heterogeneous databases, then the same company might be denoted by different constants x and x' in p and q respectively, making it impossible to join p and q in the usual way.

In general, most databases contain many domains in which the individual constants correspond to entities in the real world; examples of such "name domains" include course numbers, personal names, company names, movie names, and place names. Most previous work in data integration either assumes these "name domains" to be global, or else assumes that local "name constants" can be mapped into a global domain by some relatively simple normalization process. However, examination of real-world information sources reveals many cases in which creating a global domain by normalization is difficult. In general, the mapping from "name constants" to real entities can differ in subtle ways from database to database, making it difficult to determine if two name constants are coreferent (*i.e.*, refer to the same entity). For instance, in two Web databases listing educational software companies, we find the names "Microsoft" and "Microsoft Kids": do these denote the same company, or not? Which pairs of the following names correspond to the same research institution: "AT&T Bell Labs," "AT&T Labs," "AT&T Labs—Research," "AT&T Research," "Bell Labs," and "Bell Telephone Labs"? As these examples indicate, determining if two name constants are coreferent is often far from trivial. Frequently it requires detailed knowledge of the world, the purpose of the user's query, or both.

At first glance, developing a general data integration method for databases that lack common object identifiers might seem to be a hopeless task. Note, however, that this sort of semantic heterogeneity is not an issue in integration of unstructured textual information—in collection fusion, any set of documents can be queried in a uniform way. Note further that many Web-accessible databases present their information to the end-user through a veneer of ordinary language; for instance, business databases of the type described above would certainly include the English names of the companies involved. This raises an intriguing question: can statistical IR methods be used to resolve the lack of common object identifiers? In particular, is it possible to use the English names themselves as keys?

Constructing a database management system (DBMS) that embodies this approach raises a number of technical problems. A solution to these

problems, however, would be a great practical interest: while few pairs of distributed databases will share a common formal language for describing entities, many Web-accessible databases do use English as a common (informal) description for entities. If informal textual descriptions of entities can be used to join heterogeneous relations, then the scope of current data integration methods could be greatly extended.

In the remainder of this paper, we first describe a logic for database integration called WHIRL (for Word-based Heterogeneous Information Representation Language—the phrase "information representation language" indicating an intermediate point between information retrieval systems and knowledge representation systems). Like a conventional knowledge representation system or DBMS, WHIRL allows structured data to be represented. However, WHIRL retains the original local names, and reasons explicitly about the *similarity* of pairs of names, using statistical measures of document similarity that have been developed in the information retrieval community. As in conventional database systems, the answer to a user's query is a set of tuples; however, these tuples are ordered so that the "best" answers are presented to the user first. WHIRL considers tuples to be "better" when the name similarity conditions required by the user's query are more likely to hold.

We next describe an efficient query algorithm for WHIRL. Semantically, WHIRL is much like earlier probabilistic or "fuzzy" database logics [Fuhr 1995; Barbara et al. 1992]; however, certain properties of text make efficient inference a bit trickier. In particular, it is typically the case that many pairs of names will be weakly similar, but few will be strongly similar; this leads to inefficiencies for probabilistic inference algorithms that compute all tuples with nonzero probability. Our query-answering algorithm is novel in that it finds the highest-scoring answer tuples without generating all low-scoring tuples. The query-answering algorithm also makes heavy use of IR indexing methods.

Finally, we evaluate the algorithm experimentally on real-world data extracted from the Web. We show that our algorithm is much faster than naive inference methods, even for short queries. We also show that the inferences of the system are surprisingly accurate. In one case WHIRL's performance equals the performance of a hand-constructed, domain-specific normalization routine. In a second case, WHIRL's performance gives better performance than matching on a plausible global domain. WHIRL's performance is also robust with respect to various deficiencies in the data. This makes it possible to use WHIRL to join relations with incompatible schemas. WHIRL can also accurately join relations if local names have not been completely extracted from the surrounding text.

2. SEMANTICS OF THE WHIRL QUERY LANGUAGE

2.1 The Data Model

Recall that our motivation is to be able to integrate information from structured information sources that contain textual information. We will

thus make the natural assumption that all data are stored in relations, but that the primitive elements of each relation are fragments of text, rather than character strings or numbers. We call this data model STIR (for Storing Texts In Relations).

To represent text fragments, we adopt the widely used *vector space model* [Salton 1989], which we will now briefly review. We assume a vocabulary T of *terms*, which will be treated as atomic; terms might include words, phrases, or word stems (morphologically derived word prefixes). A fragment of text is represented as *document vector*: a 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 $\vec{v}_t$.

A number of schemes have been proposed for assigning weights to terms. We found it convenient to adopt the widely used TF-IDF weighting scheme with unit length normalization. Assuming that the document represented by $\vec{v}$ is a member of a document collection C, define $\vec{v}_t$ to have the value zero if t is not present in the document represented by $\vec{v}$, and otherwise the value $\vec{v}_t = (\log(TF_{\vec{v},t} + 1) \cdot \log(IDF_t)$, where the "term frequency" $TF_{\vec{v},t}$ is the number of times that term t occurs in the document represented by $\vec{v}$, and the "inverse document frequency" IDF_t is $|C|/n_t$, where n_t is the number of documents in C that contain the term t. The collection C associated with a document vector $\vec{v}$ will (usually) be the set of text fragments appearing in the same column of the same relation as $\vec{v}$.

The *similarity* of two document vectors $\vec{v}$ and $\vec{w}$ is given by the formula

$$sim(\vec{v}, \vec{w}) = \sum_{t \in T} \frac{\vec{v}_t \cdot \vec{w}_t}{\|\vec{v}\| \cdot \|\vec{w}\|}$$

which is usually interpreted as the cosine of the angle between $\vec{v}$ and $\vec{w}$. Notice that $sim(\vec{v}, \vec{w})$ is always between zero and one.

The general idea behind this scheme is that the magnitude of the component $\vec{v}_t$ is related to the "importance" of the term t in the document represented by $\vec{v}$. Two documents are similar when they share many "important" terms. The standard TF-IDF heuristic of assigning higher weights to terms that occur infrequently in a collection is a very reasonable one in our context: in a collection C of company names, for instance, common terms like "Inc." and "Ltd." would have low weights; uniquely appearing terms like "Lucent" and "Microsoft" would have high weights; and terms of intermediate frequency like "Acme" and "American" would have intermediate weights.[1]

An *extensional database* (*EDB*) consists of a term vocabulary T and set of relations $\{p_1, \ldots, p_n\}$. Associated with each relation p is a set of tuples

[1]Notice that this representation ignores all information about word order; thus the two strings "Cohen, William W." and "William W. Cohen" would be mapped to identical vectors. The vector space model can be extended to include some word-order information (*e.g.*, Fagan [1989]); however, in our experience, word order is seldom necessary to distinguish between object names.

tuples(p). Every tuple $\langle \vec{v}_1, \ldots, \vec{v}_k \rangle \in$ *tuples*(p) has exactly k components, and each of these components $\vec{v}_i$ is a text fragment, represented as a document vector over T. We will also assume that a *score* is associated with every tuple in p. This score will always be between zero and one, and will be denoted *score*($p\langle \vec{v}_1, \ldots, \vec{v}_k \rangle$). Informally, this score measures the degree of belief in a fact. In most applications, the score of every tuple in a base relation will be one; however, it will be convenient to allow nonunit scores, so that materialized views can be stored.

2.2 Conjunctive Queries Over Relations of Documents

WHIRL is a query language for accessing STIR relations. A *conjunctive WHIRL query* is written $B_1 \wedge \ldots \wedge B_k$ where each B_i is a *literal*. There are two types of literals. An *EDB literal* is written $p(X_1, \ldots, X_k)$ where p is the name of an EDB relation, and the X_i's are *variable symbols* (or simply *variables*). A *similarity literal* is written $X \sim Y$, where X and Y are variables; intuitively, this will be interpreted as a requirement that documents X and Y be similar. We will henceforth assume that if X appears in a similarity literal in a query Q, then X also appears in some EDB literal in Q.

Example 1. To return to the example of the introduction, the join of the relations p and q might be approximated by the query

Q_1: p(Company1,Industry) $\wedge$ q(Company2,WebSite) $\wedge$ Company1$\sim$Company2

Note that this is different from an equijoin of p and q, which could be written p(Company,Industry) $\wedge$ q(Company,WebSite). To find Web sites for companies in the telecommunications industry one might use the query:

Q_2: p(Company1,Industry) $\wedge$ q(Company2,WebSite) $\wedge$ Company1$\sim$Company2 $\wedge$ const1(IO) $\wedge$ Industry$\sim$IO

where the relation const1 contains a single document describing the industry of interest, such as "telecommunications equipment and/or services."

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

To define the semantics of WHIRL, we will extend the notion of score to single literals, and then to conjunctions. Let B be a literal, and θ a substitution such that $B\theta$ is ground. If B is an EDB literal $p(X_1, \ldots, X_k)$, we define *score*($B\theta$) = *score*($p\langle X_1\theta, \ldots, X_k\theta \rangle$) if $\langle X_1\theta, \ldots, X_k\theta \rangle \in$

tuples(p), and *score*($B\theta$) = 0 otherwise. If B is a similarity literal $X \sim Y$, we define *score*($B\theta$) = *sim*($X\theta$, $Y\theta$).

If $Q = B_1 \wedge \ldots \wedge B_k$ is a query and $Q\theta$ is ground, we define *score*($Q\theta$) = $\prod_{i=1}^{k}$ *score*($B_i\theta$). In other words, we score conjunctive queries by combining the scores of literals as if they were independent probabilities.

This combination method has some unfortunate properties; for instance, two logically equivalent queries (like B and $B \wedge B$) can have different scores. (This problem is shared by the semantics for disjunction, described below.) More generally, similarity scores are not independent probabilities, so there is no reason to expect this combination method to be in any sense optimal. However, this combination method is at least simple and relatively well-understood, and is in our view a reasonable starting point for research on this sort of data integration system.

Recall that the answer to a conventional conjunctive query is the set of ground substitutions that make the query "true" (*i.e.*, provable against the EDB). In WHIRL, the notion of provability has been replaced with the "soft" notion of score: substitutions with a high score are intended to be better answers than those with a low score. It seems reasonable to assume that users will be most interested in seeing the high-scoring substitutions, and will be less interested in the low-scoring substitutions. We formalize this as follows. Given an EDB, we define the *full answer set* S_Q for a conjunctive query Q to be the set of all θ_i such that $Q\theta_i$ is ground and has a nonzero score. We define an *r-answer* R_Q for a conjunctive query Q to be an ordered list of r substitutions $\theta_1, \ldots, \theta_r$ from the full answer set S_Q such that

—for all $\theta_i \in R_Q$ and $\sigma \in S_Q - R_Q$, *score*($Q\theta_i$) $\geq$ *score*($Q\sigma$) and
—for all $\theta_i, \theta_j \in R_Q$ where $i < j$, *score*($Q\theta_i$) $\geq$ *score*($Q\theta_j$).

In other words, R_Q contains r highest-scoring substitutions, ordered by nonincreasing score.[2]

We will assume the output of a query-answering algorithm given the query Q will *not* be a full answer set, but rather an r-answer for Q, where r is a parameter fixed by the user. To motivate the notion of an r-answer, observe that in typical situations the full answer set for WHIRL queries will be very large. For example, the full answer set for the query Q_1 given as an example above would include all pairs of company names Company1, Company2 that both contain the term "Inc". This set might be very large. Indeed, if we assume that a fixed fraction $1/k$ of company names contain the term "Inc", and that p and q each contain a random selection of n company names, then one would expect the size of the full answer set to contain $(n/k)^2$ substitutions simply due to the matches on the term "Inc"; further the full answer set for the join of m relations of this sort would be of size at least $(n/k)^m$.

[2]Note that R_Q is not always unique, since ties in score can be broken by any method.

To further illustrate this point, we computed the pairwise similarities of two lists p and q of company names,[3] with p containing 1163 names, and q containing 976 names. Although the intersection of p and q appears to contain only about 112 companies, over 314,000 name pairs had nonzero similarity. In this case, the number of nonzero similarities can be greatly reduced by discarding a few very frequent terms like "Inc". However, even *after* this preprocessing, there are more than 19,000 nonzero pairwise similarities—more than 170 times the number of correct pairings. This is due to a large number of moderately frequent terms (like "American" and "Airlines") that cannot be safely discarded.

In conclusion, it is in general impractical to compute full answer sets for complex queries, and antisocial to present them to a user. This is why we formalize the goal of query-answering to be generation of an r-answer.

In some cases, it is not appropriate to arbitrarily limit the size of the answer; instead, one would like to find all answers with scores above a certain threshold ϵ. We define an ϵ-*answer* R_Q for a conjunctive query Q to be all substitutions from the full answer set S_Q with a score of at least ϵ, ordered by nonincreasing score. The parameter ϵ provides an alternative way of limiting the number of answers to a query.

2.3 Unions of Conjunctive Queries

The scoring scheme given above for conjunctive queries can be fairly easily extended to more expressive languages. Below we consider one such extension, which corresponds to projections of unions of conjunctive queries.

A *basic WHIRL clause* is written $p(X_1, \ldots, X_k) \leftarrow Q$, where Q is a conjunctive WHIRL query that contains all of the X_i's. A *basic WHIRL view* $\mathcal{V}$ is a set of basic WHIRL clauses with heads that have the same predicate symbol p and arity k. Notice that by this definition, all the literals in a clause body are either EDB literals or similarity literals—in other words, the view is "flat," involving only extensionally defined predicates. (However, one can easily extend these semantics to all of nonrecursive Datalog, by assuming that "nonflat" views are "unfolded" before they are evaluated.) Now, consider a ground instance $a = p(\vec{x}_1, \ldots, \vec{x}_k)$ of the head of some view clause. We define the *support of* a (*relative to the view* $\mathcal{V}$ *and a given EDB*) to be the set of triples $\langle A \leftarrow Q, \theta, s \rangle$ satisfying these conditions:

(1) $(A \leftarrow Q) \in \mathcal{V}$;
(2) $A\theta = a$, and $Q\theta$ is ground; and
(3) $score(Q\theta) = s$, and $s > 0$.

The support of a will be written $support(a)$. We then define the score of $\langle \vec{x}_1, \ldots, \vec{x}_k \rangle$ in p as follows:

$$score(p\langle \vec{x}_1, \ldots, \vec{x}_k \rangle) = 1 - \prod_{\langle C,\theta,s \rangle \in support(p(\vec{x}_1, \ldots, \vec{x}_k))} (1 - s) \qquad (1)$$

[3]These lists are the relations HooverWeb and Iontech from Table IV.

As motivation for this formula, note that it is a dual of multiplication: if e_1 and e_2 are independent probabilistic events with probability p_1 and p_2 respectively, then the probability of $(e_1 \wedge e_2)$ is $p_1 \cdot p_2$, and the probability of $(e_1 \vee e_2)$ is $1 - (1 - p_1)(1 - p_2)$. We can now define the *materialization of the view* $\mathcal{V}$ to be a relation with name p which contains all tuples $\langle \vec{x}_1, \ldots, \vec{x}_k \rangle$ such that $score(\langle \vec{x}_1, \ldots, \vec{x}_k \rangle \in p) > 0$.

Unfortunately, while this definition is natural, there is a difficulty with using it in practice. In a conventional setting, it is easy to materialize a view of this sort, given a mechanism for solving a conjunctive query. In WHIRL, we would prefer to assume only a mechanism for computing r-answers to conjunctive queries. However, since Eq. (1) involves a support set of unbounded size, it appears that r-answers are not enough to even score a single ground instance a.

Fortunately, however, low-scoring substitutions have only a minimal impact on the score of a. Specifically, if $\langle C, \theta, s \rangle$ is such that s is close to zero, then the corresponding factor of $(1 - s)$ in the score for a is close to one. One can thus approximate the score of Eq. (1) using a smaller set of high-scoring substitutions, such as those found in an r-answer for large r (or an ϵ-answer for small ϵ).

In particular, let $\mathcal{V}$ contain the clauses $A_1 \leftarrow Q_1, \ldots, A_n \leftarrow Q_n$; let $R_{Q_1}, \ldots, R_{Q_n}$ be r-answers for the Q_i's; and let $R = \cup_i R_{Q_i}$. Now define the *r-support for a from R* to be the set

$$\{\langle A \leftarrow Q, \theta, s \rangle : \langle A \leftarrow Q, \theta, s \rangle \in support(a) \text{ and } \theta \in R\}.$$

Also define the *r-score for a from R* by replacing $support(a)$ in Eq. (1) with the r-support set for a. Finally, define the *r-materialization of $\mathcal{V}$ from R* to contain all tuples $\vec{x}_1, \ldots, \vec{x}_k$ with nonzero r-score, with the score of $\vec{x}_1, \ldots, \vec{x}_k$ in p being its r-score from R. We define *ϵ-support*, *ϵ-score*, and *ϵ-materialization* analogously, replacing the r-answers for the Q_i's with ϵ-answers.

Clearly, the r-materialization of a view can be constructed using only an r-answer for each clause body involved in the view. As r is increased, the r-answers will include more and more high-scoring substitutions, and the r-materialization will become a better and better approximation to the full materialized view. An analogous statement holds for an ϵ-materialization as ϵ is decreased.

Thus given an efficient mechanism for computing r-answers (or ϵ-answers) for conjunctive views, one can efficiently approximate the answers to more complex queries.

2.4 Relation to Other Logics

At the level described so far, WHIRL is closely related to earlier formalisms for probabilistic databases. In particular, if similarities were stored in a relation sim(X, Y) instead of being computed "on-the-fly," and certain

irredundancy assumptions are made,[4] then WHIRL is a strict subset of Fuhr's probabilistic Datalog [Fuhr 1995]. There are also close connections to existing formalisms for probabilistic relational databases [Barbara et al. 1992].

Given this, it might well be asked why it is necessary to introduce a new and more restricted probabilistic logic. Our response is that the assumptions made in WHIRL enable relatively efficient inference, without making the logic too restricted to handle its intended task—integration of heterogeneous, autonomous databases by reasoning about the similarity of names. In particular, these restrictions make it possible to generate an r-answer for conjunctive queries efficiently, even if the full answer set is large, and even if the document vectors used to represent local entity names are quite diverse. These claims will be substantiated more fully in Section 5 below.

3. THE QUERY PROCESSING ALGORITHM

3.1 Overview of the Algorithm

The current implementation of WHIRL implements the operations of finding an r-answer to a conjunctive query and an r-materialization of a view. In this section we will describe an efficient strategy for constructing an r-answer to a query, and then present some detailed examples of the algorithm. It should be emphasized that, while the semantics of Section 2 can be easily extended to include other sorts of similarity metrics, the specific query-answering algorithm that we have implemented in WHIRL relies heavily on the details of the document vector representation for text. In particular, the algorithm makes heavy use of "inverted indices" (which map a term to a document containing that term), and hence requires a term-based representation for text. To a lesser extent, the algorithm also relies on the facts that similarity is defined as the inner product of two vectors, and that rare terms are given heavier weights in a document vector. We leave open the problem of efficiently implementing more general similarity logics.

We begin our description of the implementation with a short overview of the main ideas used in the algorithm. In WHIRL, finding an r-answer is viewed as an optimization problem; in particular, the query-processing algorithm uses a general method called A* search [Nilsson 1987; Pearl 1984; Korf 1993] to find the highest-scoring r substitutions for a query. Viewing query processing as search is natural, given that the goal is to find a *small* number of *good* substitutions, rather than all satisfying substitutions; the search method we use also generalizes certain "short-cut" tech-

[4]Specifically, if one assumes that queries $B_1 \wedge \ldots B_k$ are "irredundant" in the sense that there is no ground substitution θ with nonzero score such that $B_i\theta = B_j\theta$ for $i \neq j$, and make the same independence assumptions made in Fuhr's Datalog$_{PID}$, then the score for a WHIRL predicate is exactly the probability of the corresponding compound event, which is the same as the probability computed by Datalog$_{PID}$.

niques used in IR ranked retrieval [Turtle and Flood 1995]. However, using search in query processing is unusual for database systems, which more typically use search only in optimizing a query; in WHIRL, search is used to generate each tuple in an answer.

To motivate our use of search, consider finding an r-answer to the WHIRL query

$$\mathsf{insiderTip(X)} \wedge \mathsf{publiclyTraded(Y)} \wedge \mathsf{X} \sim \mathsf{Y}$$

where the relation publiclyTraded is very large, but the relation insiderTip is very small. In processing the corresponding equijoin insiderTip(X) $\wedge$ publiclyTraded(Y) $\wedge$ X=Y with a conventional database system, one would first construct a query plan: for example, one might first find all bindings for X, and then use an index to find all values Y in the first column of publiclyTraded that are equivalent to some X. It is tempting to extend such a query plan to WHIRL, by simply changing the second step to find all values Y that are similar to some X.

However, this natural extension can be quite inefficient. Imagine that insiderTip contains the vector $\vec{x}$, corresponding to the document "Armadillos, Inc". Due to the frequent term "Inc", there will be many documents Y that have nonzero similarity to $\vec{x}$, and it will be expensive to retrieve all of these documents Y and compute their similarity to $\vec{x}$.

One way of avoiding this expense is to start by retrieving a *small* number of documents Y that are likely to be highly similar to $\vec{x}$. In this case, one might use an index to find all Y's that contain the rare term "Armadillos". Since "Armadillos" is rare, this step will be inexpensive, and the Y's retrieved in this step must be somewhat similar to $\vec{x}$. (Recall that the weight of a term depends inversely on its frequency, so rare terms have high weight; and hence these Y's will share at least one high-weight term with X.) Conversely, any Y' *not* retrieved in this step must be somewhat *dissimilar* to $\vec{x}$, since such a Y' *cannot* share with $\vec{x}$ the high-weight term "Armadillos". This suggests that if r is small, and an appropriate pruning method is used, a subtask like "find the r documents Y that are most similar to $\vec{x}$" might be accomplished efficiently by the subplan of "find all Y's containing the term 'Armadillos'."

Of course, this subplan depends on the vector $\vec{x}$. To find the Y's most similar to the document "The American Software Company" (in which every term is somewhat frequent) a very different type of subplan might be required. The observations suggest that query processing should proceed in small steps, and that these steps should be scheduled dynamically, in a manner that depends on the specific document vectors being processed.

In the query-processing algorithm described below, we will use the A* algorithm to search through a space of partial substitutions: for example, one state in the search space for the query given above would correspond to the substitution that maps X to $\vec{x}$ and leaves Y unbound. The steps we take through this search space are small ones, as suggested by the discussion above; for instance, one operation is to select a single term t and use an inverted index to find plausible bindings for a single unbound variable.

Generic A∗ search

procedure A∗(r,ϵ)

1. Set OPEN := $\{s_0\}$
2. While OPEN$\neq \emptyset$:
   (a) Remove s from OPEN: $s = \text{argmax}_{s' \in \text{OPEN}} f(s')$
   (b) If $f(s) < \epsilon$, exit.
   (c) If goalState(s), then output $\langle s, f(s)\rangle$, and exit if at least r answers have been output so far.
   (d) Otherwise, add children(s) to OPEN.

Using A∗ to implement WHIRL

Initial state s_0: $\langle Q, \emptyset, \emptyset\rangle$

goalState($\langle Q, \theta, E\rangle$): true iff $Q\theta$ is ground

children($\langle Q, \theta, E\rangle$):
  explode($\langle Q, \theta, E\rangle$) if a singleton p is unbound;
  otherwise, constrain($\langle Q, \theta, E\rangle$) if possible;
  otherwise, explode($\langle Q, \theta, E\rangle$)

constrain($\langle Q, \theta, E\rangle$) = all valid states in
∗ $\{\langle Q, \theta, E \cup \langle t, Y\rangle\rangle\} \cup$
  $\{\langle Q, \theta \cup \{Y_1 = \vec{v_1}, \ldots, Y_k = \vec{v_k}\}, E\rangle :$
    $\langle \vec{v_1}, \ldots, \vec{v_k}\rangle \in \text{index}(t, p, \ell)\}$

explode($\langle Q, \theta, E\rangle$) = all valid states in
  $\{\langle Q, \theta \cup \{Y_1 = \vec{v_1}, \ldots, Y_k = \vec{v_k}\}, E\rangle :$
    $\langle \vec{v_1}, \ldots, \vec{v_k}\rangle \in \textit{tuples}(p)\}$
  where $p(Y_1, \ldots, Y_k)$ is unbound in $Q\theta$

$f(\theta, E) = \prod_{B_i\theta \text{ ground}} g(B_i, \theta, E) \cdot \prod_{B_i\theta \text{ not ground}} h(B_i, \theta, E)$
where
  $g(B_i, \theta, E) = score(B_i\theta)$
∗ $h(X \sim Y, \theta, E) = \sum_{t' \in T : \langle t', Y\rangle \notin E} \vec{x}_{t'} \cdot \text{maxweight}(t', p, \ell)$
  $h(B_i, \theta, E) = 1$ if $B_i\theta$ is not a constraining literal $X \sim Y$

Fig. 1. A generic version of A∗ search, and an implementation of WHIRL based on A∗. (In lines marked ∗, $X \sim Y$ is constraining in $Q\theta$ with generator $p(Y_1, \ldots, Y_k)$ and generation index ℓ, and t is a term with nonzero weight in $X\theta$.)

Finally, we allow the search algorithm to order these operations dynamically, focusing on those partial substitutions that seem to be most promising, and effectively pruning partial substitutions that cannot lead to a high-scoring ground substitution.

3.2 A∗ search

A∗ search (summarized in Figure 1) is a graph search method which attempts to find the highest-scoring path between a given *start state* s_0 and a *goal state* [Nilsson 1987; Korf 1993]. Goal states are defined by a goalState predicate. The graph being searched is defined by a function children(s), which returns the set of states directly reachable from state s. To conduct the search the $A*$ algorithm maintains a set OPEN of states that might lie on a path to some goal state. Initially OPEN contains only the start state s_0. At each subsequent step of the algorithm, a single state is removed from the OPEN set; in particular, the state s that is "best" according to a *heuristic function*, $f(s)$, is removed from OPEN. If s is a goal state, then this state is output; otherwise, all children of s are added to the OPEN set. The search continues until r goal states have been output, or all states s in OPEN have $f(s) < \epsilon$, or the search space is exhausted.

The procedure described above is a variant of the A∗ procedure normally studied, but it has similar desirable properties, as shown in Section 4.

3.3 The Operators and Heuristic Function

We will now explain how this general search method has been instantiated in WHIRL. In processing queries, the following data structures will be used. An *inverted index* will map terms $t \in T$ to the tuples that contain them: specifically, we will assume a function index(t, p, i) which returns the set of tuples $\langle \vec{v}_1, \ldots, \vec{v}_i, \ldots, \vec{v}_k \rangle$ in $tuples(p)$ such that t appears in $\vec{v}_i$. This index can be evaluated in linear time (using an appropriate data structure) and precomputed in linear time from the EDB. We will also precompute the function maxweight(t, p, i), which returns the maximum value of $\vec{v}_t$ over all documents $\vec{v}$ in the ith column of p.

The states of the graph searched will be triples $\langle Q, \theta, E \rangle$, where Q is the query to be answered; θ is a substitution; and E is a set of *exclusions*. Goal states will be those for which $Q\theta$ is ground, and the initial state s_0 is $\langle Q, \emptyset, \emptyset \rangle$. An *exclusion* is a pair $\langle t, Y \rangle$ where t is a term and Y is a variable. Intuitively, it means that the variable Y must not be bound to a document containing the term t. More formally, a set of exclusions restricts possible states as follows:

Definition 1 (Valid State). Let Q be a conjunctive WHIRL query; let θ be a substitution; and let E be a set of pairs $E = \{\langle t_1, Y_1 \rangle, \langle t_2, Y_2 \rangle, \ldots,\}$. A state $\langle Q, \theta, E \rangle$ is valid if $\forall \langle t, Y \rangle \in E$, the document vector $Y\theta$ does not contain the term t—*i.e.*, if $\forall \langle t, Y \rangle \in E$, $(Y\theta)_t = 0$.

Below, we will define the search space so that all descendents of a node $\langle Q, \theta, E \rangle$ must be valid. This step eliminates certain redundancies the graph defined by the children function.

We will adopt the following terminology. Given a substitution θ and query Q, a similarity literal $X \sim Y$ is *constraining for* $Q\theta$ iff exactly one of $X\theta$ and $Y\theta$ are ground. Without loss of generality, we assume that $X\theta$ is ground, and $Y\theta$ is not. For any variable Y, the EDB literal of Q that contains Y is the *generator* for Y; the position ℓ of Y within this literal is Y's *generation index*. We will assume that in the query Q, each variable in Q appears exactly once in an EDB literal; thus the generator for every Y is unique.[5]

Children are generated in two ways: by *exploding* a state, or by *constraining* a state. *Exploding* a state corresponds to picking all possible bindings of some unbound EDB literal. To explode a state $s = \langle Q, \theta, E \rangle$, pick some EDB literal $p(Y_1, \ldots, Y_k)$ such that all the Y_i's are unbound by θ, and then construct all valid states of the form $\langle Q, \theta \cup \{Y_1 = \vec{v}_1, \ldots, Y_k = \vec{v}_k\}, E \rangle$ such that $\langle \vec{v}_1, \ldots, \vec{v}_k \rangle \in tuples(p)$. These are the children of s.

The second operation of *constraining* a state implements a sort of sideways information passing. To *constrain* a state $s = \langle Q, \theta, E \rangle$, pick

[5]This restriction is made innocuous by an additional predicate eq(X, Y) which is true when X and Y are bound to the same document vector. The implementation of the eq predicate is relatively straightforward, and will be ignored in the discussion below.

some constraining literal $X \sim Y$ and some term t with nonzero weight in the document $X\theta$ such that $\langle t, Y\rangle \notin E$. Let $p(Y_1, \ldots, Y_k)$ be the generator for the (unbound) variable Y, and let ℓ be Y's generation index. Two sets of child states will now be constructed. The first is a singleton set containing the state $s' = \langle Q, \theta, E'\rangle$, where $E' = E \cup \{\langle t, Y\rangle\}$. Notice that by further constraining s', other constraining literals and other terms t in $X\theta$ can be used to generate further plausible variable bindings. The second set S_t contains all valid states $\langle Q, \theta_i, E\rangle$ such that $\theta_i = \theta \cup \{Y_1 = \vec{v}_1, \ldots, Y_k = \vec{v}_k\}$ for some $\langle \vec{v}_1, \ldots, \vec{v}_k\rangle \in \mathsf{index}(t, p, \ell)$. The states in S_t thus correspond to binding Y to some vector containing the term t. The set $\mathsf{children}(s)$ is $S_t \cup \{s'\}$.

Given the operations above, there will typically be many ways to "constrain" or "explode" a state. In the current implementation of WHIRL, a state is always constrained using the pair $\langle t, Y\rangle$ such that $\vec{x}_t \cdot \mathsf{maxweight}(t, p, \ell)$ is maximal (where p and ℓ are the generator and generation index for Y). States are always exploded using the EDB relation containing the fewest tuples. A state will always be exploded if there is some unbound EDB literal corresponding to a singleton relation; if no such EDB literal exists, but there are constraining literals, then the state will be constrained; and if there are no constraining literals either, then the state will be exploded.

It remains to define the heuristic function. As shown in Section 4, the correctness of the algorithm requires that $f(\langle Q, \theta, E\rangle) = score(Q\theta)$ if θ is ground; and if θ is not ground, $f(\langle Q, \theta, E\rangle)$ must be an upper bound on $score(Q\theta')$ for all ground $\theta' \supset \theta$. We thus define

$$f(\langle Q, \theta, E\rangle) \equiv \prod_{B_i\theta \text{ ground}} g(B_i, \theta, E) \cdot \prod_{B_i\theta \text{ not ground}} h(B_i, \theta, E)$$

where $g(B_i, \theta, E) = score(B_i\theta)$, and $h(B_i, \theta, E)$ is an appropriate upper bound on $score(B_i\theta')$. We will let this bound equal 1 for all literals with the exception of *constraining literals*. For constraining literals, $h(\cdot)$ is defined as follows:

$$h(B_i, \theta, E) \equiv \sum_{t \in T : \langle t, Y\rangle \notin E} \vec{x}_t \cdot \mathsf{maxweight}(t, p, \ell) \tag{2}$$

where p and ℓ are the generator and generation index for Y.

3.4 Additional Details

In the current implementation of WHIRL, the terms of a document are stems produced by the Porter stemming algorithm [Porter 1980]. In general, the term weights for a document $\vec{v}_i$ are computed relative to the collection C of all documents appearing in the ith column of p. However, the TF-IDF weighting scheme does not provide sensible weights for relations that contain only a single tuple. (These relations are used as a means

of introducing "constant" documents into a query.) Therefore weights for these relations must be calculated as if they belonged to some other collection C'.

To set these weights, every query is checked before invoking the query algorithm to see if it contains any EDB literals $p(X_1, \ldots, X_k)$ for a singleton relation p. If one is found, the weights for the document $\vec{x}_i$ to which a variable X_i will be bound are computed using the collection of documents found in the column corresponding to Y_i, where Y_i is some variable that appears in a similarity literal with X_i. If several such Y_i's are found, one is chosen arbitrarily. If X_i does not appear in any similarity literals, then its weights are irrelevant to the computation.

The current implementation of WHIRL keeps all indices and document vectors in main memory, and consists of about 5500 lines of C and C++.[6]

3.5 Examples of WHIRL

We will now walk through some examples of this procedure. For clarity, we will assume that terms are words.

Example 2. Consider the query

const1(IO) $\wedge$ p(Company,Industry) $\wedge$ Industry$\sim$IO

where const1 contains the single document "telecommunications services and/or equipment." The first step in answering this query will be to explode the singleton relation const1. This will produce one child, s_1, containing the appropriate binding for IO, which will be placed on the OPEN list.

Next s_1 will be removed from the OPEN list. Since Industry$\sim$IO is now a constraining literal, a term from the bound variable IO will be picked, probably the relatively rare stem "telecommunications." The inverted index will be used to find all tuples $\langle co_1, \vec{ind}_1 \rangle, \ldots, \langle \vec{co}_n, \vec{ind}_n \rangle$ such that $\vec{ind}_i$ contains the term "telecommunications", and n child substitutions that map Company$=\vec{co}_i$ and Industry$=\vec{ind}_i$ will be constructed. Since these substitutions are ground, they will be given $f(\cdot)$ values equal to their actual scores when placed on the OPEN list. A new state s'_1 containing the exclusion $\langle$telecommunications,Industry$\rangle$ will also be placed on the OPEN list. Note that $f(s'_1) < f(s_1)$, since the best possible score for the constraining literal Industry$\sim$IO can match at most only four terms: "services", "and", "or", "equipment", all of which are relatively frequent, and hence have low weight.

Next, a state will again be removed from the OPEN list. It may be that $f(s'_1)$ is less than the $f(\cdot)$ value of the best goal state; in this case, a ground substitution will be removed from OPEN, and an answer will be output. Or it may be that $f(s'_1)$ is higher than the best goal state, in which case it will be removed and a new term, perhaps "equipment", will be used to generate

[6]Although it would have been preferable to implement both STIR and WHIRL using MIX [Knuth 1975].

some additional ground substitutions. These will be added to the OPEN list, along with a state s''_1 which has a larger exclusion set and thus a lower $f(\cdot)$ value.

This process will continue until r documents are generated. Note that it is quite likely that low-weight terms such as "or" will not be used at all.

In a survey article, Turtle and Flood [1995] review a number of query optimization methods for ranked retrieval IR systems. The most effective of these was one they call the *maxscore* optimization. The behavior of WHIRL on queries of the sort shown above is identical to the behavior of an IR system using the *maxscore* optimization.

Example 3. Consider the query

p(Company1,Industry) ∧ q(Company2,WebSite) ∧ Company1∼Company2

In solving this query, the first step will be to explode the smaller of these relations. Assume that this is p, and that p contains 1000 tuples. This will add 1000 states $s_1, \ldots, s_{1000}$ to the OPEN list. In each of these states, Company1 and Industry are bound, and Company1∼Company2 is a constraining literal. Thus each of these 1000 states is analogous to the state s_1 in the preceding example.

However, the $f(\cdot)$ values for the states $s_1, \ldots, s_{1000}$ will not be equal. The value of the state s_i associated with the substitution θ_i will depend on the maximum possible score for the literal Company1∼Company2, and this will be large only if the high-weight terms in the document Company1θ_i appear in the company field of q. As an example, a one-word document like "3Com" will have a high $f(\cdot)$ value if that term appears (infrequently) in the company field of q, and a zero $f(\cdot)$ value if it does not appear; similarly, a document like "Agents, Inc" will have a low $f(\cdot)$ value if the term "agents" does not appear in the first column of q.

The result is that the next step of the algorithm will be to choose a *promising* state s_i from the OPEN list—a state that could result in a good final score. A term from the Company1 document in s_i—say "3Com"—will then be picked and used to generate bindings for Company2 and WebSite. If any of these bindings results in perfect match, then an answer can be generated on the next iteration of the algorithm.

In short, the operation of WHIRL is somewhat similar to time-sharing 1000 simpler queries on a machine for which the basic unit of computation is to access a single inverted index. However, WHIRL's use of the $f(\cdot)$ function will schedule the computation of these queries in an intelligent way: queries unlikely to produce good answers can be discarded, and low-weight terms are unlikely to be used.

Example 4. Consider the query

p(Company1,Industry) ∧ q(Company2,WebSite) ∧ Company1∼Company2
∧ const1(IO) ∧ Industry∼IO

where the relation const1 contains the single document, "telecommunications and/or equipment." In solving this query, WHIRL will first explode

const1 and generate a binding for IO. The literal Industry~IO then becomes constraining, so it will be used to pick bindings for Company1 and Industry using some high-weight term, perhaps "telecommunications".

At this point there will be two types of states on the OPEN list. There will be one state s' in which only IO is bound, and ⟨telecommunications,Industry⟩ is excluded. There will also be several states $s_1, \ldots, s_n$ in which IO, Company1, and Industry are bound; in these states, the literal Company1~Company2 is constraining. If s' has a higher score than any of the s_i's, then s' will be removed from the OPEN list, and another term from the literal Industry~IO will be used to generate additional variable bindings.

However, if some s_i literal has a high $f(\cdot)$ value then it will be taken ahead of s'. Note that this is possible when the bindings in s_i lead to a good actual similarity score for Industry~IO as well as a good potential similarity score for Company1~Company2 (as measured by the $h(\cdot)$ function). If an s_i is picked, then bindings for Company2 and WebSite will be produced, resulting in a ground state. This ground state will be removed from the OPEN list on the next iteration only if its $f(\cdot)$ value is higher than that of s' and all of the remaining s_i's.

This example illustrates how bindings can be propagated through similarity literals. The binding for IO is first used to generate bindings for Company1 and Industry, and then the binding for Company1 is used to bind Company2 and Website. Note that bindings are generated using high-weight, low-frequency terms first, and low-weight, high-frequency terms only when necessary.

4. CORRECTNESS OF THE IMPLEMENTATION

We will now show formally that the implementation of WHIRL described in Section 3 implements the semantics described in Section 2.

THEOREM 1. *Let WHIRL(r, ϵ, Q) be the A* algorithm, as instantiated in Figure 1. Then*

—*the output of WHIRL(r, ϵ, Q) is an ϵ-answer for Q, if $r = +\infty$, and*
—*the output of WHIRL(r, ϵ, Q) is an r-answer for Q, if $\epsilon = 0$ (and the graph G contains at least r goal states).*

PROOF. The proof of the theorem proceeds in three steps.

For the first step, define a graph G to be a *bounded tree* if it is a tree of finite depth and finite branching factor in which the goal states are all leaves. We will show that the graph defined by the children function is a bounded tree.

For the next step, define a heuristic function $f(\cdot)$ to be *admissible* iff for all states s and all states s' reachable from s, $f(s) \geq f(s')$. We will show that $f(\langle Q, \theta, E \rangle)$ is admissible.

Finally, will we argue that the A* algorithm has this property: if $f(\cdot)$ is admissible and the graph G defined by the children function is a bounded

tree, then the A* variant of Figure 1 outputs in nonincreasing order the goal states with the largest $f(\cdot)$ values.

The termination conditions for A* are clear. Notice also that each state $s = \langle Q, \theta, E \rangle$ encodes a substitution θ, and that $f(s) = score(Q\theta)$ for θ that are ground for Q. Thus the three claims above imply that the algorithm of Figure 1 will compute an ϵ-answer for Q when called with $r = +\infty$, and an r-answer for Q when called with $\epsilon = 0$ (assuming that G contains at least r goal states).

To see that children defines a bounded graph, note first that either exploding a state and constraining a state produces a finite number of children. The number of ways a state can be exploded or constrained is immaterial, since for each state, only one of these operations will be chosen (following the heuristics described in Definition 1). Also, a state can be exploded at most once for every EDB literal, and if $X \sim Y$ is a constraining literal in which X is bound to a document with w nonzero terms, then the state can be constrained (using this literal) at most w times. This bounds the depth of the graph.

To see that children defines a tree, let desc(s) denote the descendent states of s, and consider two sibling states s_i and s_j with common parent $s = \langle Q, \theta, E \rangle$. If s_i and s_j were formed by exploding s, then clearly desc(s_i) and desc(s_i) are disjoint (since some variables will be bound to different values). If s_i and s_j were formed by constraining s, then there are two cases to consider. On one case, both s_i and s_j are in the set S_t, shown in Definition 1 to be the valid elements of

$$\{\langle Q, \theta \cup Y_1 = \vec{v_1}, \ldots, Y_k = \vec{v_k}\}, E\rangle : \langle \vec{v}_1, \ldots, \vec{v}_k \rangle \in \mathsf{index}(t, p, \ell))\}.$$

Again, clearly desc(s_i) and desc(s_i) are disjoint. In the other case, exactly one of s_i, s_j is in S_t, and the other is not: let us assume without loss of generality that $s_j = s' = \langle Q, \theta, E \cup \{\langle t, Y \rangle\}\rangle$ and $s_i \in S_t$. In this case, the descendents of s_j must be disjoint from the descendents of s_i, since (because of the exclusion $\langle t, Y \rangle$) no descendents of s_j can bind Y to a vector with nonzero weight for t, and all of the descendents of s_i bind Y some vector with nonzero weight for t. Thus the graph generated by the children function is a tree.

To see that $f(\cdot)$ is admissible, it is sufficient to note that Eq. (2) is an upper bound on the score of $B_i\theta'$ relative to any ground superset θ' of θ associated with a valid state.

Finally, we wish to show that if $f(\cdot)$ is admissible, and the graph G defined by the children function is a bounded tree, then algorithm A* of Figure 1 outputs in nonincreasing order the goal states with the largest $f(\cdot)$ values. This statement is a slight (and unsurprising) variant[7] of the correctness property traditionally associated with A* search [Nilsson 1987],

[7]The principle differences are that A* usually is considered to minimize a sum of costs, rather than maximizing a score that is a product, and that A* usually is only used to find a single "best" goal state. The former difference is trivial; the latter, somewhat more important.

but we include a proof for the sake of completeness. Specifically, we will show the following:

> Let s_i be some goal state output by A*, and let i be the number of times the "while" loop had been executed when s_i was output. Then for any goal state s in G, if $f(s) > f(s_i)$, then s was output by A* at some iteration j, where $j < i$.

Let $P(s, k)$ be the proposition "s, or some ancestor of s, is in OPEN at iteration k." Since s_i selected at stage i has maximal value of all nodes in OPEN, it clearly cannot be the case that s is in OPEN at iteration i. Further, by the admissibility of f, we have that $f(s') \geq f(s) > f(s_i)$ for any ancestor s' of s, so it also cannot be the case that any ancestor of s is in OPEN at iteration i. Hence $P(s, i)$ is false.

Let j be the largest number for which $P(s, j)$ is true. Note that $j < i$ and $j \geq 0$; since all goal states are reachable from s_0, $P(s, 0)$ is true. At iteration j, either s or an ancestor of s was removed from OPEN, and neither s nor an ancestor of s was inserted on OPEN. Let s_j be the node removed at iteration j. If s_j were a nongoal state (an ancestor of s) then some child of s' of s_j is also an ancestor of s, and s' would be inserted on OPEN, making $P(s, j + 1)$ true. Thus s_j must be the goal state, s, and $s_j (= s)$ will be output at iteration j.

This concludes the proof of correctness of the algorithm. □

5. EXPERIMENTAL RESULTS

We evaluated our implementation of WHIRL along two dimensions. First, we wished to measure the time needed to evaluate queries. Second, we wished to measure the accuracy of the answers produced by WHIRL. In this evaluation we used the measures of precision and recall traditionally used in the statistical IR community. All experiments were performed using an implementation of WHIRL that keeps all indices and document vectors in main memory.

5.1 Controlled Timing Experiments

We evaluated run-time performance with CPU time measurements on a specific class of queries, which we will henceforth call *similarity joins*. A *similarity join* is a query of the form

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

An r-answer to this query will consist of the r tuples from p and q such that X_i and Y_j are most similar. In these experiments we used the relations described in Table I.

Similarity join queries have several advantages for benchmarking purposes. This query type is highly relevant to our research goals, since it is directly related to the sort of data integration problem which led us to develop WHIRL. This class of queries is also sufficiently constrained in form so that it can be handled using simple algorithms built on top of

Table I. Relations Used in Controlled Timing Experiments

Name and Description	#Tuples
`IMDB` - movie names	31,281
`VideoFlicks` - movie names	37,572
`Hoovers` - company names	2,474
`ReutersTrain` - news stories	13,625
`ReutersTest` - news stories	6,188

well-known, previously existing IR search methods. This makes it possible to compare the query optimizations used in WHIRL with previous query optimizations. In particular, we will compare WHIRL with the following algorithms:

—The *naive method for similarity joins* takes each document in the ith column of relation p in turn, and submits it as an IR ranked retrieval query to a corpus corresponding to the j-column of relation q. The top r results from each of these IR queries are then merged to find the best r pairs overall. This might be more appropriately called a "semi-naive" method; on each IR query, we use inverted indices, but we employ no special query optimizations.

—As noted above, WHIRL is closely related to *maxscore* optimization [Turtle and Flood 1995]. We thus compared WHIRL to a *maxscore method for similarity joins*; this method is analogous to the naive method described above, except that the *maxscore* optimization is used in finding the best r results from each "primitive" query.

The version of the interpreter used here is also slightly different from the one used in other experiments. To facilitate this comparative study, we used a version of WHIRL which shares as much low-level code as possible with the implementations for the naive and maxscore methods; elsewhere, we used a version of the WHIRL interpreter that was extended to better support experimentation. The two implementations are identical at the level of description given in Section 3.

To see how these algorithms behave, we used them to compute the top 10 answers[8] for the similarity join of subsets of the IMDB and VideoFlicks relations. In particular, we joined size n subsets of both relations, for various values of n between 2000 and 30,000. The results for the movie domain 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. Note that the absolute time required to compute the join is fairly modest—with $n = 30{,}000$, WHIRL takes well under a minute[9] to pick the best 10 answers from the 900 million possible candidates.

[8]In other experiment (not reported here) we have explored the result of increasing r up to several thousand. For these sorts of problems the compute time for WHIRL grows no worse than linearly with r.

[9]Timing results are given in CPU seconds on a MIPS Irix 6.3 with 200MHz R10000 processors.

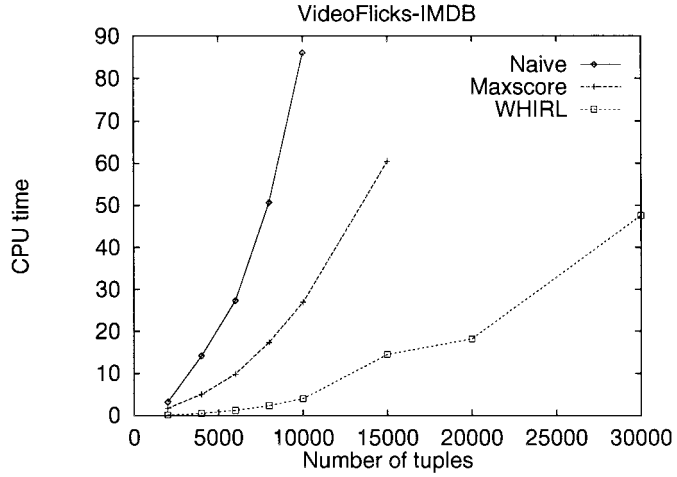

Fig. 2. Runtime in CPU seconds for similarity joins of movie names.

We also joined ReutersTrain and Hoovers using the company name column of Hoovers and the story column of ReutersTrain. This application of similarity joins corresponds to searching for all references in the Reuters corpus to any company listed in Hoovers, and illustrates an interesting blending of IR search with data integration. The results are shown in the first graph of Figure 3. On these problems the *maxscore* method does not improve over the naive method with respect to CPU time.[10] However, WHIRL speeds up the naive method by a factor of 2–4. The absolute time required is again small—about 5 CPU seconds for $n = 2474$.

It should be noted that the run-time for these queries is fast in part because some of the documents being joined are names. Names tend to be short and highly discriminative, and thus behave more like traditional database keys than arbitrary documents might. This point is illustrated experimentally in the second graph of Figure 3, which shows the run-time for similarity joins of ReutersTrain with ReutersTest. This is again a plausible task: it represents using a similarity join for a kind of duplicate detection. Although WHIRL still improves substantially over its nearest competitor the absolute time requirements are much higher: WHIRL takes nearly four minutes to find the 10 most similar documents with $n = 3000$. In this case none of the columns involved in the join contain short, namelike documents.

5.2 Timing Results for Typical Queries

The similarity joins studied in the previous section are important because they are the simplest WHIRL queries which cannot be answered by either a conventional database system, or a conventional IR ranked retrieval sys-

[10]It does, however, greatly reduce the number of accesses to the inverted index, as Turtle and Flood observed.

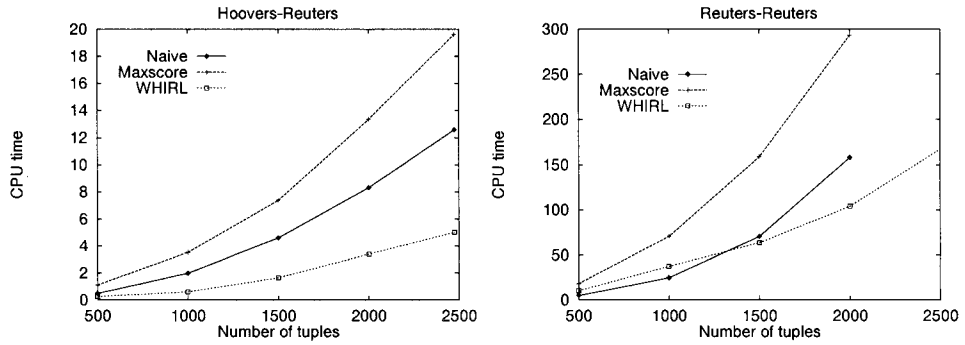

Fig. 3.. Runtime in CPU seconds for similarity joins of company names and news stories.

tem. However, these simple queries are probably not typical of the sort of queries that one would like to pose to a real data integration system; one would expect that typical user queries would be more selective, and more complex.

To better understand WHIRL's behavior on "typical" queries, WHIRL was embedded into a working, Web-based, data integration system [Cohen 1998b]. This system spiders a number of related Web sites and extracts a WHIRL knowledge base, which can then be queried. The main additional components of this system are an HTTP server interface to WHIRL, which allows conjunctive queries[11] to WHIRL to be easily formulated, and a spider program, which downloads and extracts data from HTML pages. Two moderately large domains were implemented for this system, one integrating information on birds of North America, and one integrating information about educational computer games.

The interface to the system in the game domain allows the user to ask a question by filling out an HTML form—*e.g.*, "help me find reviews of games that are in the category 'art', are recommended by two or more sites, and are designed for children six years old." This question is then translated into a conjunctive WHIRL query. The interface to the bird domain is similar: an example of a question that might be posed in this domain, again using a forms interface, is "help me find pictures of birds in the order pelicaniforms that have been sighted in New Jersey and are endangered or threatened." In addition to a forms interface for constructing complex questions, the bird domain interface also supports browsing the database, and a "quick search" feature, in which a simple keyword query can be used to search relevant portions of the database. Browsing and "quick search" are implemented by translating browsing commands and simple keyword searches into appropriate WHIRL queries.

[11]The user's queries are conjunctive, but not necessarily flat—they may involve WHIRL views, which are used in this system to make the different data sources more compatible.

Table II. Performance of the WHIRL Interpreter on Real-World Queries

	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

We made both domains available on the Web, and recorded each query issued to the system. Later we took a "snapshot" of each domain[12] 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, 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.

A comparative study of performance is inappropriate here, since arbitrary WHIRL queries cannot be answered by any means other than the algorithm of Section 3; thus, Table II simply summarizes the results. Note that the average response time is well under a second.

Table III provides some additional detail on these results. For each domain and for each number k, we show the number of queries that are k-way joins, the average number of similarity literals used in k-way join queries, and the average time to execute the k-way join queries.[13] In these samples, many of the queries are relatively simple, but a substantial fraction are moderately complex: in the bird domain, about a quarter are 4-, 5-, or 6-way joins, and in the game domain, about a quarter are 5- or 6-way joins. WHIRL is still quite efficient, even on the longer queries.

5.3 Average Precision of Similarity Joins

Efficient reasoning is only worthwhile if the inferences made are useful ones. Therefore, we also evaluated the *accuracy* of inferences made by WHIRL, again using data taken from the Web.

We adopted the following methodology. Again focusing on similarity joins, 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 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

[12]Notice that since the sites indexed are not static, the size of the databases changes every time the spiders are run.

[13]The number k does not count "joins" with singleton relations like the const relations in the example queries of Section 3.5.

Table III. A More Detailed Summary of the Performance of the WHIRL Interpreter on Real-World Queries

Domain	k	# k-way Joins	Avg #Sim Literals	Average Time
Birds	≤ 2	47	2.0	0.02
	3	22	3.3	0.03
	4	14	3.8	0.35
	5	4	3.8	1.90
	6	4	5.0	0.22
Games	≤ 2	35	1.4	0.06
	3	20	3.9	0.08
	4	16	4.1	0.50
	5	23	5.3	0.26
	6	6	6.0	1.61

standard IR system. In particular, we measured the quality of a ranking using (*noninterpolated*) *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 germane, 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 kth answer is correct and letting $c(k) = 0$ otherwise, average precision is the quantity $\Sigma_{k=1}^{r} c(k) \cdot a_k/k$.

Note that average precision is 1 only when all correct answers precede all incorrect answers. In the experiments below, we used r-answers of size $r = 1000$ to compute average precision.

To evaluate similarity joins, we picked 13 pairs of relations from several different domains. Table IV summarizes the relations used in these experiments. In the movie domain, we used movie names as a primary key, and as a secondary key, we used a special key constructed by a hand-coded normalization procedure for film names developed for an application of the Information Manifold,[14] another data integration system [Levy et al. 1996b]. In the animal and bird domains, for several pairs of relations, we used common names as the primary key, and scientific names as a secondary key, again with a hand-coded matching procedure.[15] In the business domain, we joined Iontech and HooversWeb using company names as the primary key, and the string representing the "site" portion of the home page as a secondary key; also in the business domain, we joined two

[14]Thanks to Alon Levy and Jaewoo Kang for providing me with the data and normalization routines.

[15]In the BirdCall relation, we manually cleaned the secondary keys—but *not* the primary keys—by fixing spelling errors.

Table IV. Relations Used in Measuring Accuracy of Similarly Joins

Name and Description	Primary/Secondary Keys	#Tuples
`MovieLink` - movie listings	movie name/normalized name	78
`Review` - movie reviews	movie name/normalized name	421
`IntFact1` - animal factsheets	common name/URL of factsheet	11
`IntFact2` - animal factsheets	common name/URL of factsheet	23
`IntFact3` - animal factsheets	common name/URL of factsheet	16
`SWFact` - animal factsheets	common name/URL of factsheet	52
`FWSFact` - animal factsheets	common name/ URL of factsheet	27
`NMFSFact` - animal factsheets	common name/ URL of factsheet	78
`Endanger` - endangered species	common name/scientific name	990
`ParkAnim` - animals found	common name/scientific name	4719
`IntBird1` - pictures of birds	common name/URL of image file	15
`IntBird2` - pictures of birds	common name/URL of image file	155
`DonBirdPic` - pictures of birds	common name/URL of image file	23
`MBRBirdPic` - pictures of birds	common name/URL of image file	564
`IntBirdMap` - maps of bird	common name/URL of map file	20
`BirdMap` - maps of bird	common name/URL of map file	317
`BirdCall` - bird calls	common name/scientific name	68
`BirdList` - birds of North America	common name/scientific name	914
`HooverWeb` - large employers	name of business/URL of home page	1163
`Iontech` - hi-tech companies	name of business/URL of home page	976
`Fodor` - restaurants	restaurant listing/manual key	532
`Zagrat` - restaurants	restaurant listing/manual key	331
`Demo` - on-line demos	name of computer game/manual key	113
`AgeList` - appropriate ages for games	name of computer game/manual key	1626
`IntPark` - US parks	name of park/URL of home page	258
`Park` - US Parks	name of park/URL of home page	398

collections of restaurant "listings" (containing a name, address, phone number, and brief description of the cuisine served), using manually constructed secondary keys.[16] In the domain of computer games, we joined two lists of computer games, again using manually constructed secondary keys.

We also used several pairs of relations which were derived from Web sites in which someone had manually collected a large number of pointers to external pages of a specific type. Such "hotlists" are common on the Web, but we restricted ourselves to hotlists with the following properties: the pages were associated with named objects (*e.g.*, animals, birds, or national parks); the external site contained an index of pages of this type; and in the hotlist site, pages had been systematically relabeled. In this situation, the names associated with the pages can be used as a primary key, and the URLs of the pages can be used as a secondary key.

For example, one such page collected pointers to animal "factsheets" from three different sites, associating each factsheet with an animal name. We took the animal names from this page and split them into three relations (IntFact1, IntFact2, and IntFact3) based on the external site hosting the

[16]Thanks to Sheila Trejada for supplying the restaurant data.

Table V. Average Precision for Similarity Joins

Domain	Relations Joined	Average Precision
Movies	MovieLink/Review	100.0%
Animals	IntFact1/SWFact	100.0%
	IntFact2/FWSFact	99.6%
	IntFact3/NMFSFact	97.1%
	Endanger/ParkAnim	95.2%
Birds	IntBirdPic1/DonBirdPic	100.0%
	IntBirdPic2/MBRBirdPic	99.1%
	IntBirdMap/BirdMap	91.4%
	BirdCall/BirdList	95.8%
Businesses	Fodor/Zagrat	99.5%
	HooverWeb/Iontech	84.9%
National Parks	IntPark/Park	95.7%
Computer Games	Demo/AgeList	86.1%

associated URL. We then constructed relations based on the index pages in the external sites (SWFact, FWSFact, NMFSFact) and joined the appropriate pairs of relations. The relations IntBird1, IntBird2, IntBirdMap, and IntPark were derived from similar sources. The results are summarized in Table V.

On these domains, similarity joins are extremely accurate. In the movie domain, the performance is actually identical to the hand-coded normalization procedure. Overall, the average value for average precision is more than 95%. These results contrast with the typical performance of statistical IR systems on retrieval problems, where the average precision of a state-of-the art IR system usually is closer to 50% than 90%. This suggests that the similarity reasoning required to match names is easier than the similarity reasoning required to process a typical IR ranked retrieval query.

In the experiments, we used the secondary key as a "gold standard"; however, in some of the domains, the matching procedure for the secondary keys is somewhat error prone. This is especially true for Web sites used as secondary keys in joining HooverWeb and Iontech. To estimate the accuracy of the secondary keys we took the top 100 pairs in the join of HooverWeb and Iontech, and manually checked all pairings marked as "incorrect" according to the secondary key. Table VI shows some representative pairs of names for this particular problem, together with an indication as to whether the pairing is correct (√) or incorrect (×). Of the 13 pairings marked "incorrect," there were 11 in which the secondary keys were wrong, one in which the WHIRL pairing was wrong (at rank 77), and one pair where correctness could not be easily determined. This suggests that the similarity join is actually *more* accurate than the use of Web sites as a key.

Performance in the computer game domain also seems to be anomolous—average precision here is much lower than for other domains with reliable secondary keys. The problem here seems to be that computer game names vary more widely than in the other domains; also, there are many groups of related, but distinct, games with similar names. For example, "Disney's Ready for Math with Pooh" and "Disney's Ready to Read with Pooh" denote

Table VI. Pairs of Names from the Hoovers and Iontech Relations

√	Texas Instruments Incorporated	TEXAS INSTRUMENTS INC
√	The New York Times Company	NEW YORK TIMES CO
√	Campo Electronics, Appliances and Computers, Inc.	CAMPO ELECTRONICS APPLIANCES
√	Cascade Communications Corp.	CASCADE COMMUNICATION
√	The McGraw-Hill Companies, Inc.	MCGRAW-HILL CO
√	U S WEST Communications Group	U S WEST INC
×	Silicon Valley Group, Inc.	SILICON VALLEY RESEARCH INC
×	The Reynolds and Reynolds Company	REYNOLDS & REYNOLDS CO
√	InTime Systems International, Inc.	INTIME SYSTEMS INTERNATIONAL I

different games, but "Disney's Animated StoryBook, 101 Dalmations" and "101 Dalmations—Disney Interactive" both denote the same game. As another example, "The Lion King: Storybook" and "Lion King Animated StoryBook" both denote the same game, and "Disney's Activity Center, The Lion King" and "The Lion King Activity Center" both denote a second, distinct game. Soft matches based on similarity are necessarily less accurate in such a domain.[17]

5.4 Similarity Joins with Incompatible Schemata

Another problem that occurs in integrating heterogeneous data is the problem of incompatible schemata. For example, consider trying to associate professors with their university affiliation using the relations professor(name, workAddress) and university(name, state). These relations cannot be joined in any conventional sense; however, it is plausible that concatenating a university name and state would give a text fragment similar to a workAddress (although not an identical fragment, since typically a workAddress would have a number of extra terms, such as "Department of Computer Science", in addition to some variant of the university name and its state). In this case, an appropriate similarity join might give a useful result, even though the objects being joined are in fact different.

We explored this possibility in the following experiments. We began by considering different schemata for the MovieLink and Review relations, with the aim of constructing problems that are similar to the sort of incompatible-schemata problem given above, but still possible to evaluate rigorously by checking individual pairings. The full schemas for these relations are MovieLink(movieName, cinemaName, address, phone, zipcode) and Review(movieName, newspaper, review) respectively. For MovieLink, we considered a variation in which each tuple contains a "movie listing"—*i.e.*, a single document containing a movie name plus a complete cinema address. For Review, we considered a variation in which each tuple contains only a review entry, and no separate movie name field; thus similarity joins must compute the similarity of a movie name or movie listing to the full text of a

[17]On the other hand, normalization of these types of names would be extremely difficult.

movie review.[18] We then computed similarity joins with each possible combination of a MovieLink variant and a Review variant.

One would expect the irrelevant "noise" words that appear along with the movie names to have some adverse affect on precision. In our experiments with the Review and Movielink relations, however, the effect was quite slight: joining movie names to movie listings reduces average precision by only a little over 1%, and joining movie listings to complete reviews reduces average precision by less than 6%. Finally, joining movie listings to movie names leads to no measurable loss in average precision.

To substantiate these results, we considered two additional domains. The Web page associated with the relation demo lists on-line demos of educational games for children as a list of free-text descriptions, some representative items of which 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).*

In the experiments of Table V, game names were (manually) extracted from this listing. Here, we used each complete list item from Demo as a key, and joined these list items with the game names from AgeList. We recorded a pairing as correct if the AgeList game was mentioned in the Demo list item. In this difficult domain, the addition of these "noise words" decreases average precision from 86.1% to 67.1%. Although "noise words" have a greater effect here, it should be noted that even without extraction, the result of the similarity join is certainly accurate enough to be usable; for instance, the first 23 pairings contain only two mistakes, appearing at ranks 5 and 6, both of which incorrectly pair the games "Mario Teaches Typing" and "Mario Teaches Typing 2."

A final experiment used the BirdCall relation. This relation was originally derived by manually extracting bird names from a Web site devoted to bird calls. The bird names appear in paragraph-length descriptions of related groups of sound files, some examples[19] of which follow:

—*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)*

—*American Goldfinch Carduelis tristis. Song in flight (47kB) "as he goes he sings with a thin, wiry voice Per-CHIC-o-ree, and he does so rhythmically with his undulating flight, always breaking out with the song just at the*

[18]The movie reviews usually contain a title naming the movie being reviewed, but also contain a lot of additional text. The average length of a review is more than 400 words.

[19]Punctuation has been added for clarity. The text in the original is also structured by interleaved graphics, and by positioning on the page.

Table VII. Average Precision for Similarity Joins between Pairs of Relations with Incompatible Schemas, or Incompletely Extracted Names

Fields Joined	Average Precision
`MovieLink` movie name/`Review` movie name	100.0%
`MovieLink` movie listing/`Review` movie name	100.0%
`MovieLink` movie name/full review	98.8%
`MovieLink` movie listing/full review	94.6%
`Demo` game name/`AgeList` game name	86.1%
`Demo` paragraph/`AgeList` game name	67.1%
`BirdCall` paragraph/`BirdList` bird name	95.8%
`BirdCall` paragraph/`BirdList` bird name	83.0%

crest of the wavelike curve." (Mathews) Limekiln Lake, August 1996. Song in flight (62kB); Perching song (59kB); and another (48kB) from the same bird. Long Island, July 1997.

Using a methodology similar to that used with the Demo relation, we joined these descriptions to a list of bird names. Average precision for the data with manually extracted names was 95.8%; without extraction, average precision is decreased to 83.0%. These results are summarized in Table VII.

6. RELATED WORK

Chaudhuri et al. [1995] present efficient solutions to the problem of loosely integrating Boolean text queries with database queries. In contrast, we have considered a much tighter integration between databases and statistical IR queries. The assumptions made by Chaudhuri et al. are not particularly appropriate in the context of heterogeneous database integration.

As noted above in Section 2.4, WHIRL is closely related to probabilistic databases (*e.g.*, Fuhr [1995] and Barbara et al. [1992]). To our knowledge such database systems have not been used in data integration tasks. Furthermore, the implementation of WHIRL is unique in generating only a few "best" answers to a query; existing probabilistic database systems typically find all tuples with nonzero probability. As we argued above in Section 2.2, this would often be impractical for the problems encountered in this sort of heterogeneous database integration, due to the prevalence of weak matches between documents.

Fuzzy set theory [Zadeh 1965] has also been used as the basis for "soft" database systems [Bosc and Prade 1997]. Fagin [1998] and others have proposed algorithms that find the best few answers to a conjunctive query in this model. However, unlike WHIRL, these algorithms make assumptions about the independence of the atomic queries, and provide the best answer only with high probability.

The WHIRL query algorithm borrows heavily from techniques previously used to optimize ranked retrieval searches in statistical IR. To our knowledge, these techniques have not been previously used for approximating the join of lists of documents. More generally, the sort of approximate join

implemented in WHIRL has not been investigated in the IR literature, although numerous other hybrids of statistical IR techniques with database representations have been proposed (*e.g.*, Schäuble [1993] and Fuhr [1995]).

There has also been much work on approximate matching techniques for the removal of duplicates and merging of heterogeneous data sources [Newcombe et al. 1959; Felligi and Sunter 1969; Kilss and Alvey 1985; Huffman and Steier 1995; Hernandez and Stolfo 1995; Monge and Elkan 1997]. Most of the approximate matching methods proposed are domain-specific (*e.g.*, the Synoname™ algorithm [Borgman and Siegfried 1992] for personal names). A notable exception is the Smith-Waterman edit distance adopted by Monge and Elkan [1997]. Applying these techniques is a relatively expensive off-line process which usually is not guaranteed to find the best matches, due to the nearly universal use of "blocking" heuristics which restrict the number of similarity comparisons.

Here, we have considered approximate matching using the vector space model of similarity. This model enjoys a number of advantages. Like Smith-Waterman, it is domain-independent. It is extremely well supported experimentally as a similarity metric for text; we note that in a previous comparison, a simple term-weighting method gave better matches than the Smith-Waterman metric [Monge and Elkan 1996]. Finally, by using inverted indices, it is possible to quickly locate items similar to a given item. Exploitation of this property results in an approximate matching algorithm that is guaranteed to find the best pairings, but still fast enough to interleave with query answering. Note that interleaving matching with query answering, rather than computing the best matches off-line, has an important consequence: rather than commit early as to whether a match is correct or incorrect, one can propagate uncertainty about approximate matches, and then use the propagated uncertainty to rank answers presented to the end-user. Another advantage of interleaving matching and query answering is that, in some circumstances, incorrect matches can lead to correct inference [Cohen and Hirsh 1998].

There have also been a number of approaches to data integration which address issues orthogonal to the problem of lack of common domains. Examples of such work include "semi-structured" data models [Suciu 1996; Abiteboul and Vianu 1997; Suciu 1997]. While we have focused here on relational models, due to their simplicity, we believe that many of the basic principles of WHIRL can be applied to more complex data models as well.

A number of systems seek to provide a database-like view of the Web (*e.g.*, Fiebig et al. [1997], Mendelzon and Milo [1997], and Konopnicki and Schmueli [1995]), in which queries can express combinations of keyword searches and hypertext connectivity constraints; in effect, these languages offer a means of declaratively navigating the Web. An interesting variant of this approach is represented by the WebKB project [Craven et al. 1998], which uses machine learning techniques to find views of the Web (in the database sense) that model objects and relationships in the real world. WHIRL differs in emphasis from this work, in that we focus on answering

queries involving information stored on data-like Web pages, rather than using data about the architecture of the Web itself to answer queries, or to learn models of the world. At a more technical level, our work focuses on integrating sites that contain no explicit links connecting them. WHIRL also differs from most database-like views of the Web in that it includes statistical IR methods for searching within documents, rather than boolean keyword search methods.

In its basic motivation, our work is inspired by previous work in the integration of heterogeneous data sources, such as data sources on the Web [Levy et al. 1996b; Arens et al. 1996; Garcia-Molina et al. 1995; Atzeni et al. 1997; Tomasic et al. 1997; Bayardo et al. 1997]. None of these previous systems, however, include a "fuzzy" matching procedure for names; instead they construct global domains using hand-crafted domain-specific normalization schemes, or domain-specific matching algorithms [Fang et al. 1994].

The connection between WHIRL and other data integration systems is discussed more fully in another paper [Cohen 1998b], which describes the WHIRL-based data integration system mentioned in Section 5.2. The focus of that paper is on mechanisms for converting HTML information sources into STIR databases, and other practical issues in fielding a data integration system. In contrast, this paper focuses on efficient theorem-proving algorithms for WHIRL, and evaluation of WHIRL's performance in controlled experiments.

Some of the results of this paper have appeared elsewhere in a more preliminary form [Cohen 1997; 1998a]. Additional experiments concerning the accuracy of WHIRL's inferences on queries involving projection have also appeared elsewhere [Cohen and Hirsh 1998].

7. CONCLUSIONS

In an ideal world, one would like to integrate information from heterogeneous autonomous databases with little or no human effort. In other words, one would like data to be easily shared among databases. Unfortunately, such data sharing is difficult with current data models. One fundamental and critical problem is the lack of global domains: different databases are likely to use different constants to refer to the same real-world entity, making operations like joins across relations from different databases impossible.

We believe the data model and query language presented in this paper represent a significant advance toward the long-term goal of easily sharable data. We have outlined an approach to the integration of structured heterogeneous information sources, based on extended conventional database query languages with standard IR methods for reasoning about textual similarity. The approach is embodied in an implemented "information representation language" called WHIRL. WHIRL is intended for integration of relations that are semantically heterogeneous in the sense that there is no common naming scheme for entities.

The problem of integrating relations without global domains has received little prior attention. Current data integration systems typically use domain-specific rules to normalize entity names, and then use the normalized versions of these names as keys. These normalization rules are developed manually, sometimes at considerable effort. In practice, the cost of this process in terms of human time limits data integration systems to relatively well structured data collected from a relatively small number of sites. Furthermore, normalization is prone to error, and unlike WHIRL, a system based on normalized keys has no way of either assessing the likelihood of such errors or (more importantly) informing the user of potential errors.

Our experiments show that the accuracy of WHIRL's "similarity joins" are quite good, even compared to hand-coded integration schemes based on normalization. In one case WHIRL's performance equals the performance of a hand-constructed, domain-specific normalization routine. In a second case, WHIRL's performance gives better performance than matching on a plausible global domain. WHIRL is robust enough to join relations using incompletely extracted names. Also, WHIRL is efficient—the current implementation can handle multiple-join queries on moderate sized databases (containing a few tens of thousands of tuples) at interactive speeds.

Although these results are encouraging, many additional topics remain to be addressed. There are many well-known methods for conducting an approximate A* search; some or all of these may lead to substantial performance improvements. The current version of WHIRL handles heterogeneous data, but not in a distributed fashion; this is another intriguing topic for future work. We would also like to consider the issue of closely integrating WHIRL with appropriate learning methods for text categorization [Lewis 1992; Cohen and Singer 1996], adjusting numerical parameters for queries [Bartell et al. 1994; Boyan et al. 1994; Cohen et al. 1997], and learning logical expressions [Quinlan 1990].

ACKNOWLEDGMENTS

The author is grateful to Alon Levy for numerous helpful discussions while I was formulating this problem, and for comments on a draft of the paper; to Jaewoo Kang and Sheila Tejada, for providing data; to Alex Borgida, Sal Stolfo, and Mark Jones for comments on the paper; to Susan Cohen for proofreading; and to Edith Cohen, David Lewis, Haym Hirsh, Fernando Pereira, Divesh Srivastava, Dan Suciu, and many other colleagues for helpful advice and discussions.

REFERENCES

ABITEBOUL, S. AND VIANU, V. 1997. 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).

ARENS, Y., KNOBLOCK, C. A., AND HSU, C.-N. 1996. Query processing in the SIMS information mediator. In A. Tate 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 D. Suciu Ed., *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.

BARBARA, D., GARCIA-MOLINA, H., AND PORTER, D. 1992. The management of probabilistic data. *IEEE Transations on knowledge and data engineering 4*, 5 (October), 487–501.

BARTELL, B. T., COTTRELL, G. W., AND BELEW, R. K. 1994. Automatic combination of multiple ranked retrieval systems. In *Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1994).

BAYARDO, R. J., BOHRER, W., BRICE, R., CICHOCKI, A., FOWLER, J., HELAL, A., KASHYAP, V., KSIEZYK, T., MARTIN, G., NODINE, M., RASHID, M., RUSINKIEWICZ, M., SHEA, R., UNNIKRISHAN, C., UNRUH, A., AND WOELK, D. 1997. Infosleuth: an agent-based semantic integration of information in open and dynamic environments. In *Proceedings of the 1997 ACM SIGMOD* (May 1997).

BORGMAN, C. L. AND SIEGFRIED, S. L. 1992. Getty's Synoname and its cousins: a survey of applications of personal name-matching algorithms. *Journal of the American Society for Information Science 43*, 7, 459–476.

BOSC, P. AND PRADE, H. 1997. An introduction to the fuzzy set and possibility theory-based treatment of queries and uncertain or imprecise databases. In *Uncertainty management in information systems.* Kluwer Academic Publishers.

BOYAN, J., FREITAG, D., AND JOACHIMS, T. 1994. A machine learning architecture for optimizing web search engines. Technical Report WS-96-05, American Association of Artificial Intelligence.

CHAUDHURI, S., DAYAL, U., AND YAN, T. 1995. Join queries with external text sources: execution and optimization techniques. In *Proceedings of the 1995 ACM SIGMOD* (May 1995).

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

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

COHEN, W. W. 1998b. A Web-based information system that reasons with structured collections of text. In *Proceedings of Autonomous Agents-98* (St. Paul, MN, 1998).

COHEN, W. W. AND HIRSH, H. 1998. Joins that generalize: Text categorization using WHIRL. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (New York, NY, 1998), pp. 169–173.

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* (Zurich, Switzerland, 1996), pp. 307–315. ACM Press.

COHEN, W. W., SCHAPIRE, R. E., AND SINGER, Y. 1997. Learning to order things. In *Advances in Neural Processing Systems 10* (Denver, CO, 1997). MIT Press.

CRAVEN, M., DIPASQUO, D., FREITAG, D., MCCALLUM, A., MITCHELL, T., NIGAM, K., AND SLATTERY, S. 1998. Learning to extract symbolic knowledge from the world wide web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)* (Madison, WI, 1998).

DUSCHKA, O. M. AND GENESERETH, M. R. 1997a. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-97)* (Tucson, AZ, May 1997).

DUSCHKA, O. M. AND GENESERETH, M. R. 1997b. Query planning in infomaster. In *Proceedings of the Twelfth Annual ACM Symposium on Applied Computing (SAC97)* (San Jose, CA, February 1997).

FAGAN, J. L. 1989. The effectiveness of a nonsyntactic approach to automatic phrase indexing for document retrieval. *Journal of the American Society for Information Science 40*, 2, 115–132.

FAGIN, R. 1998. Fuzzy queries in multimedia database systems. In *Proc. 1998 ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)* (1998).

FANG, D., HAMMER, J., AND MCLEOD, D. 1994. The identification and resolution of semantic heterogeneity in multidatabase systems. In *Multidatabase Systems: An Advanced Solution for Global Information Sharing*, pp. 52–60. IEEE Computer Society Press, Los Alamitos, California.

FELLIGI, I. P. AND SUNTER, A. B. 1969. A theory for record linkage. *Journal of the American Statistical Society 64*, 1183–1210.

FIEBIG, T., WEISS, J., AND MOERKOTTE, G. 1997. RAW: a relational algebra for the Web. In D. Suciu Ed., *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.

FUHR, N. 1995. Probabilistic Datalog—a logic for powerful retrieval methods. In *Proceedings of the 1995 ACM SIGIR conference on research in information retrieval* (New York, 1995), pp. 282–290.

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)* (Naharia, Israel, November 1995).

HERNANDEZ, M. AND STOLFO, S. 1995. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD* (May 1995).

HUFFMAN, S. AND STEIER, D. 1995. 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.

KILSS, B. AND ALVEY, W. 1985. Record linkage techniques—1985. Statistics of Income Division, Internal Revenue Service Publication 1299-2-96. Available from http://www.bts.gov/fcsm/methodology/.

KNUTH, D. E. 1975. *The Art of Computer Programming, Volume I: Fundamental Algorithms (second edition)*. Addison-Wesley, Reading, MA.

KONOPNICKI, D. AND SCHMUELI, O. 1995. W3QS: a query system for the world wide web. In *Proceedings of the 21st International Conference on Very Large Databases (VLDB-96)* (Zurich, Switzerland, 1995).

KORF, R. 1993. Linear-space best-first search. *Artificial Intelligence 62*, 1 (July), 41–78.

LEVY, A. Y., RAJARAMAN, A., AND ORDILLE, J. J. 1996a. Query answering algorithms for information agents. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)* (Portland, Oregon, August 1996).

LEVY, A. Y., RAJARAMAN, A., AND ORDILLE, J. J. 1996b. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB-96)* (Bombay, India, September 1996).

LEWIS, D. 1992. Representation and learning in information retrieval. Technical Report 91-93, Computer Science Dept., University of Massachusetts at Amherst. PhD Thesis.

MENDELZON, A. AND MILO, T. 1997. 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).

MONGE, A. AND ELKAN, C. 1996. The field-matching problem: algorithm and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (August 1996).

MONGE, A. AND ELKAN, C. 1997. 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).

NEWCOMBE, H. B., KENNEDY, J. M., AXFORD, S. J., AND JAMES, A. P. 1959. Automatic linkage of vital records. *Science 130*, 954–959.

NILSSON, N. 1987. *Principles of Artificial Intelligence.* Morgan Kaufmann.

PEARL, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA.

PORTER, M. F. 1980. An algorithm for suffix stripping. *Program 14*, 3, 130–137.

QUINLAN, J. R. 1990. Learning logical definitions from relations. *Machine Learning 5*, 3, 239–266.

SALTON, G. ED. 1989. *Automatic Text Processing*. Addison Wesley, Reading, Massachusetts.
SCHÄUBLE, P. 1993. SPIDER: A multiuser information retrieval system for semistructured and dynamic data. In *Proceedings of the 1993 ACM SIGIR conference on research in information retrieval* (Pittsburgh, PA, 1993), pp. 318–327.
SUCIU, D. 1996. Query decomposition and view maintenance for query languages for unstructured data. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB-96)* (Bombay, India, 1996).
SUCIU, D. ED. 1997. *Proceedings of the Workshop on Management of Semistructured Data*. Available on-line from http://www.research.att.com/suciu/workshop-papers.html, Tucson, Arizona.
TOMASIC, A., AMOUROUX, R., BONNET, P., AND KAPITSKAIA, O. 1997. The distributed information search component (Disco) and the World Wide Web. In *Proceedings of the 1997 ACM SIGMOD* (May 1997).
TURTLE, H. AND FLOOD, J. 1995. Query evaluation: strategies and optimizations. *Information processing and management 31*, 6 (November), 831–850.
ZADEH, L. A. 1965. Fuzzy sets. *Information and Control 8*, 338–353.

Received November 1998; revised July 1999; accepted March 2000