Knowledge Integration for Structured Information Sources Containing Text (Extended Abstract)

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

August 1, 1997

Abstract

Knowledge integration is the integration of distributed, heterogeneous databases, such as those available on the World Wide Web. In this paper we will consider a new type of knowledge integration problem, namely, the problem of combining information from relations that lack common object identifiers. A general technique for this problem is proposed, based on well-studied similarity measures for text, and the observation that Web-based databases often present their information to the end user through a veneer of text. We describe an extension of Datalog called WHIRL which allows passages of ordinary text to be used as keys. WHIRL supports documents as a built-in type, similarity reasoning with a built-in predicate, and answers every query with a list of answer substutitions that are ranked according to an overall similarity score. Experiments with a prototype implementation show that WHIRL is efficient for databases with tens of thousands of tuples, and produces highly accurate results for many queries.

From the *SIGIR-97 Workshop on Networked Information Retrieval*

1 Introduction

Collection fusion is the integration of distributed text collections. An analogous problem exists for distributed structured information sources, and currently the integration of distributed, heterogeneous databases, sometimes known as "knowledge integration", is an active area of research in the database community [Duschka and Genesereth, 1997b; Levy *et al.*, 1996a; Arens *et al.*, 1996; Garcia-Molina *et al.*, 1995]. Largely inspired by the proliferation of database-like sources on the World Wide Web, this work has addressed a diverse set of issues, ranging from access to non-relational "semi-structured" information sources [Suciu, 1997], to combining databases with differing schemata [Levy *et al.*, 1996b; Duschka and Genesereth, 1997a].

In this paper we will consider a new type of knowledge integration problem, namely, the problem of combining information from relations that lack common object identifiers. As a concrete example, consider the following two information sources: a relation `p` associating companies with a short description of their industries, and a second relation `q` associating companies with web sites. It is natural to consider joining these two relations on their "company" fields. However, while the join operation is trivial to perform if both relations use the same naming scheme for companies—for instance, if they both use tickertape symbols as a key—it is not obvious how such a join can be performed in the general case. Unfortunately, in many domains, standardized key-like names for entities do not exist, or else are not widely used; for instance, in the domain of non-publically traded companies, there is no obvious field that could be used as a key.

At first glance, the task of developing a general method for joining relations that lack common object identifiers would appear to be hopeless. Note, however, that this sort of semantic heterogeneity is not a issue in collection fusion—with statistical IR methods, any set of documents containing a common set of terms can be effectively integrated. 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 a 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 knowledge integration methods could be greatly extended.

In the remainder of this paper, we will first describe a DBMS which allows small pieces of ordinary text to be used as keys. We believe that in most domains, exact matching of names will not be reliable; thus, in our model, ordinary joins are replaced with "soft joins", which pair up tuples with similar keys, rather than identical keys. This is accomplished with an extension of Datalog that includes documents as a built-in type and similarity reasoning as a built-in operation—similarity being assessed using a standard metric from information retrieval (cosine similarity in a vector-space model) [Salton, 1989]. The similarity reasoning also can be used to integrate and combine relational algebra operations with the sort of ranked retrieval that is commonly implemented in IR systems.

After describing our model, and summarizing the pertinent details of a partial implementation of it, we will present some preliminary experimental results. Our contention is that these results show this approach to be both effective and efficient on realistic databases of moderate size.

2 Foundations

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 is stored in relations, but that the primitive elements of each relation are *documents*, rather than say, character strings or numbers. We call this data model STIR (for Storing Texts In Relations).

More formally, an *extensional database (EDB)* is defined to be a set of relations $R = \{p_1, \ldots, p_n\}$, and a *vocabulary* T. The vocabulary is simply a set of *terms*, which will be treated here as atomic. Associated with each relation p is a character string $NAME(p)$[1]; a natural number $ARITY(p)$, which is a natural number; and tuple set $TUPLES(p)$. Every tuple $\langle \mathbf{v_1}, \ldots, \mathbf{v_a} \rangle \in TUPLES(p)$ has exactly a components, where $a = ARITY(p)$, and each of these components $\mathbf{v_i}$ is a *document vector*. We will assume that document vectors are represented using standard vector space techniques [Salton, 1989], and are of unit length; more formally, a document vector is simply a unit-length vector in $\mathcal{R}^{|T|}$. We will denote the component of $\mathbf{v}$ which corresponds to the term $t \in T$ by $\mathbf{v}^t$.

2.2 The retrieval language

Access to a STIR database is via an extension of Datalog [ct: something] called WHIRL (for Word-based Heterogeneous Information Retrieval Logic). For simplicity, we will describe subset of the full logic—a language for conjunctive queries to the EDB.

In WHIRL, a *(conjunctive) 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 $p(X_1, \ldots, X_a)$ 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. A WHIRL query W is *well-formed* if each variable in W appears exactly once in a EDB literal; in other words, if the variables in EDB literals are distinct from each other, and also distinct from variables appearing in other EDB literals, and if the both variables appearing in a similarity literal also appear in some EDB literal. We will henceforth assume that all WHIRL queries are well-formed.

[1]We will assume that the names of different relations are distinct, and when it is convenient, we will drop entirely the distinction between a relation and its name; for instance, if s is the name of a relation p, we may use $TUPLES(s)$ instead of $TUPLES(p)$.

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

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

To find the web sites for companies in a computer-related industry one might use the query

`p(Company1,Industry)` $\wedge$ `q(Company2,WebSite)` $\wedge$ `Company1`$\sim$`Company2`
$\wedge$ `const1(I0)` $\wedge$ `Industry`$\sim$`I0`

where the relation `const1` contains a single one-word document: "computer".

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

Given an EDB, we define the *score* of a ground substitution θ and a WHIRL query (or literal) W as follows.

- If W is an EDB literal $p(X_1, \ldots, X_a)$, then $SCORE(W, \theta) = 1$ if $\langle X_1\theta, \ldots, X_a\theta\rangle \in TUPLES(p)$, and $SCORE(W, \theta) = 0$ otherwise.
- If W is a similarity literal $X \sim Y$, then $SCORE(W, \theta) = \sum_{t \in T} \mathbf{x}^t \cdot \mathbf{y}^t$, where $\mathbf{x} = X\theta$ and $\mathbf{y} = Y\theta$. In other words, $SCORE(W, \theta)$ is the cosine similarity between the documents represented by $X\theta$ and $Y\theta$. Note that X and Y are of unit length, so $SCORE(W, \theta)$ is between 0 and 1.[2]
- If W is a WHIRL query $B_1 \wedge \ldots \wedge B_k$, then $SCORE(W, \theta) = \prod_{i=1}^{k} SCORE(B_i, \theta)$. Again note that $SCORE(W, \theta)$ is between 0 and 1.

Recall that the answer to a Datalog query is typically 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 in some sense better answers than those with a low score. Formally, given an EDB, an *answer to a WHIRL query* W is an ordered list of all pairs $\langle\theta_1, SCORE(W, \theta_1)\rangle, \ldots, \langle\theta_r, SCORE(W, \theta_r)\rangle$, such that each θ_i is a ground substitution for W with a non-zero score, and the pairs are ordered by non-increasing score—*i.e.*, for all $i < j$, $SCORE(W, \theta_i) \geq SCORE(W, \theta_j)$.

Each θ_i in this list will be referred to as an *answer substitution* for the query W. Although we define the answer to W to contain *all* substitution/score pairs with a non-zero score, in most practical contexts it would be reasonable for a theorem-prover to return a small prefix of this list—for instance, the top 10 or so answer substitutions.

We note that, if the relations in the EDB all contain distinct documents, and if only conjunctive queries are considered, the assumption of well-formedness is clearly a convenience,

[2] We use the cosine similarity function here for concreteness, but any plausible similarity function with a range of $[0, 1]$ could be used.

rather than an actual restriction. In more general settings, the well-formedness assumption could be conveniently relaxed be introducing an equality predicate $X = Y$. We note also that if all documents contain a single term, then the score of all similarity literals will be either zero or one, and the answer substitutions for a WHIRL query correspond exactly to the substitutions that satisfy the Datalog query obtained by replacing every similarity literal $X \sim Y$ with an equality literal $X = Y$.

3 Implementation

It can be shown that WHIRL, like Datalog, has polynomial data complexity; however, it seems likely that naive theorem-proving strategies would be impractically expensive for moderately large databases. We would like to make the implementation as efficient as possible, by making use of appropriate search techniques, and appropriate optimizations from the field of information retrieval. Our implementation also focuses on the problem of finding the best few answer substitutions, rather than on enumerating all of them.

There are many implementation details which affect performance, and which we hope to discuss in the long paper. Here we will focus on the search procedure, which is the most novel part of WHIRL. Briefly, a Porter stemmer is used to generate terms, and that a fairly standard TFIDF weighting scheme is used. Currently, we keep document frequency statistics independently for each column of each relation.[3]

3.1 A general search procedure

It is clear that the logic described above could be readily extended in a number of ways—for instance, a complete definite clause logic could easily be defined, or other logic programming extensions (*e.g.*, predicates for numerical calculation) could be readily imported. So far we have implemented only the relatively simple logic described above. Our implementation consists of about 5500 lines of C and C++.[4]

In the implementation, an A$*$ search procedure [Nilsson, 1987; Korf, 1993] is used to find the r top-scoring answer substitutions for each query, where r is a parameter set by the user. 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". Goal states are defined by a `goalState` predicate, and the graph being searched is defined by a function `children`(s), which returns the set of states directly reachable from 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" $h(s)$ 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 the search space is exhausted.

[3] An exception to this rule is made for relations that contain a single document, which we have been using here as a means of introducing "constant" documents into a query. The DF statistics for these documents are taken from the column to which they expected to be similar.

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

```
procedure A∗(r,s_0,goalState(·),children(·))
begin
    OPEN := {s_0}
    while (OPEN ≠ ∅) do
        s := argmax_{s'∈OPEN} h(s')
        OPEN := OPEN - {s}
        if goalState(s) then
            output ⟨s, h(s)⟩
            exit if r answers have been printed
        else
            OPEN := OPEN ∪ children(s)
        endif
    endwhile
end
```

Figure 1: A generic A∗ search procedure

The procedure we use is a variant of the A∗ procedure normally studied, but it is straightforward to show that it has similar desirable properties. In particular, define a heuristic function $h(\cdot)$ to be *admissible* iff for all states s and all states s' reachable from s', $h(s) \geq h(s')$, and define a graph G to be *bounded tree* if it is a tree in which the goal states are all leaves. It is straightforward to show that if $h(\cdot)$ is admissible and the graph G defined by the `children` function is a bounded tree containing at least r goal states, then this A∗ variant will output in non-increasing order the r goal states with the largest heuristic values.

3.2 Searching for answer substitutions

We will use the A∗ algorithm to compute the first r answer substitutions for a query W. Below we will give a brief technical description of the specifics of our approach, and then present some more detailed examples.

The states of the graph searched will be pairs $\langle\theta, E\rangle$, where θ is a substitution, and E is a set of *exclusions*. Goal states will be those for which θ is ground for W, and the initial state s_0 is $\langle\emptyset, \emptyset\rangle$. 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. Formally, we will say that a substitution θ is *E-valid* if $\forall\langle t, Y\rangle \in E$, $(Y\theta)^t = 0$. Exclusions will be used to force the graph defined by the `children` function to be a tree.

Note that the heuristic function $h(\theta, E)$ must be admissible, and must coincide with the scoring function $SCORE(W, \theta)$ on ground substitutions. We will define $h(\theta, E)$ to be $\prod_{i=1}^{k} h'(B_i, \theta, E)$, where h' will be an appropriate upper bound on $SCORE(B_i, \theta)$. We will let this bound equal $SCORE(B_i, \theta)$ for ground B_i, and let it equal 1 for non-ground B_i, with one important exception, which we will now discuss: the case of *constraining literals*.

In the discussion, we will use this terminology. Given a substitution θ and query W, a similarity literal $X \sim Y$ is a *constraining* iff exactly one of $X\theta$ and $Y\theta$ are ground; without loss of generality, I will always assume that $X\theta$ is ground and $Y\theta$ is not. For any variable Y, the EDB literal of W that contains Y is the *generator* for Y, the position ℓ of Y within this literal is Y's *generation index*, and $DOM(Y)$ denotes the set of document vectors that appear in column ℓ of p—*i.e.*, $DOM(Y) = \{\mathbf{y}_\ell : \langle \mathbf{y}_1, \ldots, \mathbf{y}_\ell, \ldots, \mathbf{y}_a \rangle \in TUPLES(p)\}$.

Readers familiar with ranked retrieval will no doubt realize that "constraining literals" $X \sim Y$ play a familiar role: the bound variable X corresponds to a query (in the information retrieval sense), and choosing a binding for Y corresponds to answering that query with document from the collection $DOM(Y)$. It is thus reasonable to suspect that effective treatment of these constraining literals will be crucial in constructing an *efficient* theorem prover for WHIRL. We will thus introduce a special case of h' for constraining literals: $h'(B_i, \theta, E) = \min(1, h''(B_i, \theta, E))$, where

$$h''(B_i, \theta, E) \equiv \sum_{t \in T : \langle t, Y \rangle \notin E} \mathbf{x}_t \cdot (\max_{\mathbf{y} \in DOM(Y)} \mathbf{y}^t)$$

Note that this is an upper bound on the score of B_i relative to any ground superset of θ that is E-valid.

It remains to define the `children(·)` function. 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 $s = \langle \theta, E \rangle$, pick some EDB literal $p(Y_1, \ldots, Y_a)$ such that all the Y_i's are unbound by θ, and then construct all states of the form $\langle \theta \cup \{Y_1 = \mathbf{v_1}, \ldots, Y_a = \mathbf{v_a}\}, E \rangle$ such that $\langle \mathbf{y}_1, \ldots, \mathbf{y}_a \rangle \in TUPLES(p)$ and θ_i is E-valid. These are the children of s.

To *constrain* a state $s = \langle \theta, E \rangle$, pick some constraining literal $X \sim Y$ and some term t with non-zero weight in the document $X\theta$. Let $p(Y_1, \ldots, Y_a)$ be the generator for the (unbound) variable Y, and let ℓ be Y's generation index. We use an special inverted index for T over the set $DOM(Y)$ to find the set S_t of tuples in p that contain the term t in the ℓ-th column:

$$S_t = \{\langle \mathbf{v_1}, \ldots, \mathbf{v}_\ell, \ldots, \mathbf{v}_a \rangle : \langle \mathbf{v_1}, \ldots, \mathbf{v}_\ell, \ldots, \mathbf{v}_a \rangle \in DOM(Y) \text{ and } {\mathbf{v}_\ell}^t > 0\}$$

Two sets of child states will now be constructed. The first is a singleton set containing the state $s' = \langle \theta, E \cup \{\langle t, Y \rangle\}\rangle$.[5] The second set contains all states $\langle \theta \cup \{Y_1 = \mathbf{v_1}, \ldots, Y_a = \mathbf{v_a}\}, E \rangle$ such that $\{\langle \mathbf{y}_1, \ldots, \mathbf{y}_a \rangle \in S_t\}$ and θ_i is E-valid. The set `children(`s`)` is the union of these two sets.

Given the operations above, there will typically be many ways to "constrain" or "explode" a state. In the current implementation, a state is always constrained using the term/variable pair t, Y such that $\mathbf{x}_t \cdot (\max_{\mathbf{y'} \in DOM(Y)} \mathbf{y'}^t)$ is maximal. States are exploded only if there are no constraining literals, and then always exploded using the EDB relation containing the fewest tuples.

3.3 Examples of WHIRL

We will now walk through some examples of this procedure.

[5]Notice that by further constraining s', other constraining literals and other terms t in $X\theta$ can be used to generate plausible variable bindings.

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". With $\theta = \emptyset$, there are no constraining literals, so first step in answering this query will be to explode the smallest relation, in this case `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 term "`telecommun*`". The inverted index will be used to find all tuples $\langle \mathbf{co}_1, \mathbf{ind}_1\rangle, \ldots, \langle \mathbf{co}_n, \mathbf{ind}_n\rangle$ such that $\mathbf{ind}_i$ contains the term "`telecommun*`", and n child substitutions that map `Company=`$\mathbf{co}_i$ and `Industry=`$\mathbf{ind}_i$ will be constructed. Since these substitutions are ground, they will be given $h(\cdot)$ values equal to their actual scores when placed on the `OPEN` list. A new state s_1' containing the exclusion $\langle$telecommun*, `Industry`$\rangle$ will also be placed on the `OPEN` list. Note that $h(s_1') < h(s_1)$, since the best possible score for the constraining literal `Industry`$\sim$`IO` can match at most only four terms: "`servic*`" "`and`", "`or`", "`equip*`".

Next, a state will again be removed from the `OPEN` list. It may be that $h(s_1')$ is less than 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 $h(s_1')$ is still higher than the best goal state, in which case it will be removed and a new term, perhaps "`equip*`", will be used to generate some additional ground substitutions. These will be added to the `OPEN` list, along with a state s_1'' which has an $h(\cdot)$ value derived from the remaining three terms.

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 an actual run of this query on the `Hoovers` relation (see below), the first 40-odd documents are found using the first term "`telecommun*`", and so for $r \leq 40$ only one indexing step is needed.

In a survey article, Turtle and Flood [1995] review a large number of query optimization methods, the most effective of which is one they call the *maxscore* optimization. The behavior of WHIRL for queries of this type is much like the behavior of an IR system using the maxscore optimization. This is not a coincidence. On complex WHIRL queries, the performance bottleneck is access to the `OPEN` list. (This is implemented as a heap, and can grow extremely large.) In practice, the number of heap accesses is highly correlated with number of accesses to the inverted index; and since the performance bottleneck for most IR systems is disk access, the maxscore optimization is designed to minimize accesses to the inverted index.

Example 3 Consider the query

`p(Company1,Industry)` $\wedge$ `q(Company2,WebSite)` $\wedge$ `Company1`$\sim$`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`$\sim$`Company2` is a constraining literal. Thus each of these 1000 states is analogous to the state s_1 in the preceding example.

However, the $h(\cdot)$ values for the states $s_1, \ldots, s_{1000}$ will not be equal. The value of the state associated with the substitution θ_i will depend on the maximum possible score for the literal `Company1`$\sim$`Company2`, and this will be large only if the high-weight terms in the document `Company1`θ_i appear in the company field of `q`. As an example, a one-word document like "`3Com`" will have a high $h(\cdot)$ value if that term appears (infrequently) in the company field of `q`, and a zero $h(\cdot)$ value if it does not appear; similarly, a document like "Agents, Inc" will have a low $h(\cdot)$ value if the term "`agent*`" does not appear in the appropriate field 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 an good final score. A term from thge `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 $h(\cdot)$ 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)` $\wedge$ `q(Company2,WebSite)` $\wedge$ `Company1`$\sim$`Company2`
$\wedge$ `const1(IO)` $\wedge$ `Industry`$\sim$`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`$\sim$`IO` then becomes constraining, so it will be used to pick bindings for `Company1` and `Industry` using some high-weight term, perhaps "`telecommun*`".

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 $\langle$`telecommun`$*$, `Industry`$\rangle$ 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`$\sim$`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`$\sim$`IO` will be used to generate additional variable bindings.

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

This example illustrates how bindings can be "propogated" by the search routine 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.

Relation	Tuples	Field/Source
IMDB	31,281	1 movie name/year http://us.imdb.com
Videoflicks	37,572	1 movie name/year/genre http://www.videoflicks.com
Review	232	3 movie name, newspaper, review http://www.cinema.pgh.pa.us/movie/reviews
MovieLink	78	5 movie name, cinema name, address, phone, zipcode http://www.movielink.com
Hoovers	2,474	2 company name, industry http://www.hoovers.com
HooversWeb	1,163	3 company name, industry, website http://www.hoovers.com
Iontech	976	4 company name, website, tickertape, industry http://www.iontech.com
ReutersTrain	13,625	2 story, keywords http://www.research.att.com/~lewis/reuters21578.html
ReutersTest	6,188	2 story, keywords http://www.research.att.com/~lewis/reuters21578.html
Animal1	990	2 common name, scientific name http://endeavor.des.ucdavis.edu/nps
Animal2	4,719	3 common name, scientific name, range http://www.nceet.snre.umich.edu/EndSpp/ES.lists.html

Table 1: Relations used in the experiments

4 Experimental results

We evaluated our implementation of WHIRL along two dimensions. First, we wished to measure the time needed to evaluate typical queries, and compare this time cost with other strategies. Second, we wished to measure to effectiveness of WHIRL on the traditional performance measures of precision and recall.

4.1 Timing results

We used the relations described in Table 1 to evaluate the performance of WHIRL. Most of these relations are projections of relations that might actually be useful; we have extracted out only those fields that are seemed likely to be useful in the experiments.

We evaluated run-time performance with CPU time measurements[6] on a specific class of queries, which we will henceforth call *similarity joins*. A *similarity join* is a query of the form

[6]The current implementation of WHIRL keeps most data in main memory, so there are few disk accesses during computation.

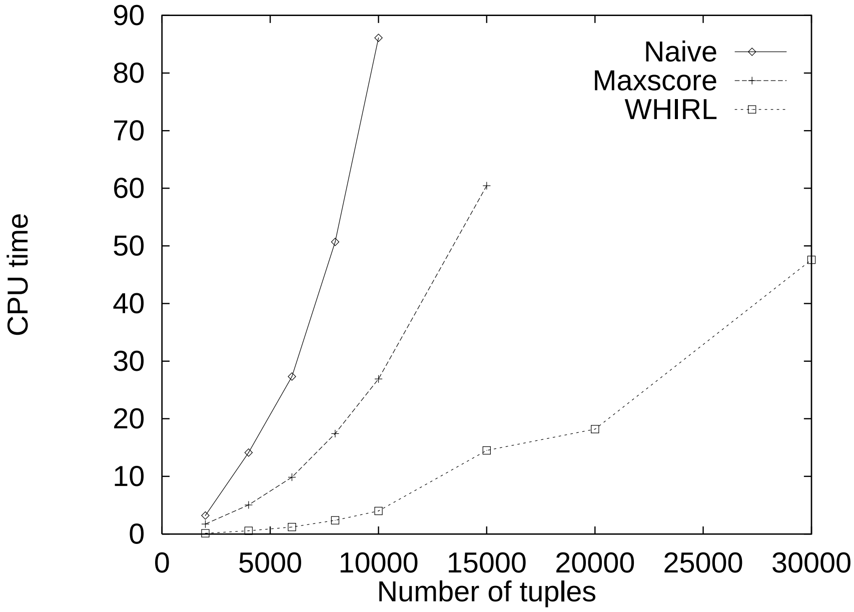

Figure 2: Runtime for similarity joins: movie domain

$\mathtt{p}(X_1, \ldots, X_i, \ldots, X_a) \ \wedge \ \mathtt{q}(Y_1, \ldots, Y_j, \ldots, Y_b) \ \wedge \ X_i \sim Y_j$

An answer to this query will consist of the r tuples from p and q such that X_i and Y_j are most similar.

This type of query has several advantages for benchmarking purposes. First, it 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. Second, they are moderately expensive to evaluate by naive methods; this means that it is possible to both measure performance differences, and to conduct systematic experiments. Finally, this class of queries it is sufficiently constrained in form so that it can be handled using simple algorithms built on top of well-known, previously existing IR search methods. This means that makes it possible to compare the query optimizations used in WHIRL with previous query optimizations. In particular, we will compare WHIRL with these algorithms.

- The *naive method for similarity joins* takes each document in the i-th field of relation p in turn, and submits it as a query to a corpus corresponding to the j-column of relation q. The top r results from each of these "primitive" queries are then merged to find the best r pairs overall. With the naive method, we use inverted indices for each relation column, but employ no other 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.

To see how these algorithms behave, we first used them to compute the top 10 answers

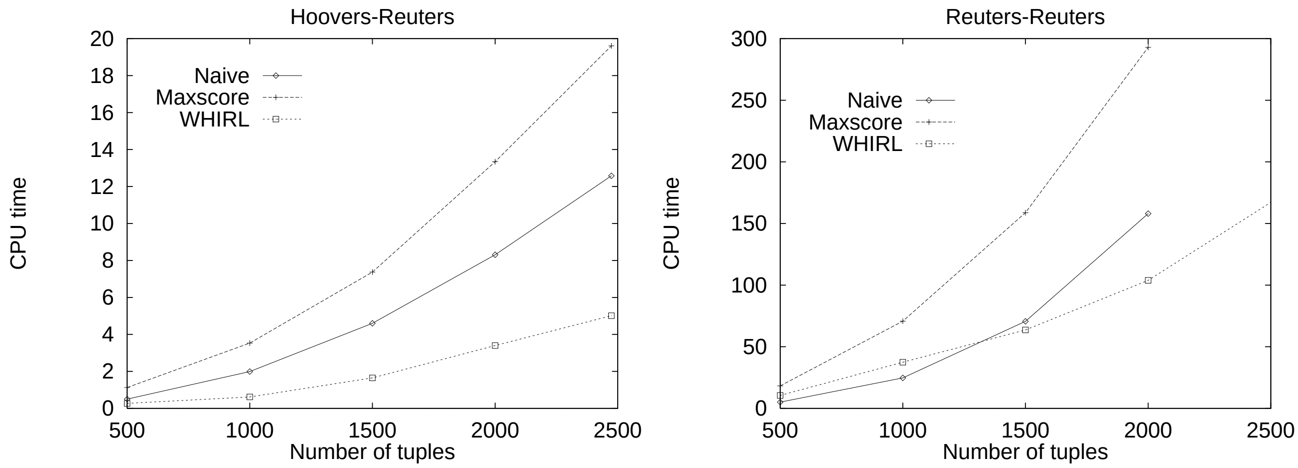

Figure 3: Runtime for similarity joins: company/news domains

for the similarity join of subsets of the IMDB and VideoFlicks relations.[7] In particular, we joined size n subsets of both relations, for various values of n between 2000 and 30,000. The results are shown in Figure 2. For this data, WHIRL speeds up the maxscore method by a factor of between 4 and 9, and speeds up the naive method by a factor of 20 or more. Note that the absolute time required to compute the join is fairly modest—with $n = 30,000$, WHIRL takes well under than a minute to pick the best 10 answers from the 900 million possible candidates.

We performed similar experiments with two other pairs of relations, joining ReutersTrain with Hoovers, and ReutersTrain with ReutersTest. We joined ReutersTrain and Hoovers using the company name field of Hoovers and the story field of ReutersTrain. This application of similarity joins corresponds to searching for all mentions in the Reuters corpus of any company listed in Hoovers, and illustrates an interesting blending of IR search with knowledge 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. (It does, however, greatly reduce the number of accesses to the inverted index, as Turtle and Flood observed.) 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 $n = 2474$.

Joining ReutersTrain with ReutersTest represents using similarity join for a sort of duplicate detection. The run-time results for this case are shown in the second graph in Figure 3. 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 $n = 3000$.

The names in the Hoovers, Videoflicks, and IMDB relations are much shorter than the Reuters stories. Our conjecture is that WHIRL is most efficient when at least one of the columns being joined contains mostly short documents. We will explore this issue in more detail in the longer paper.

[7] In the longer version of the paper, we will also report results on returning larger numbers of answers per query.

Similarity Joins

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

Constrained Similarity Joins

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

Table 2: Average precision for similarity joins

4.2 Precision-recall results

To evaluate the accuracy of the answers produced by WHIRL, we adopted the following methodology. Again focusing on similarity joins, we collected a number of pairs of relations which contained two or more plausible "key" fields. One of these fields, the "primary key", was used in the similarity literal in the join. The second key field was then used to check the correctness of proposed pairings; specifically, a pairing was marked as "correct" if the secondary keys matched (using an appropriate matching procedure) and "incorrect" otherwise. We then treated "correct" pairings in the same way that "relevant" documents are typically treated in evaluation of a ranking proposed by a standard IR system. For each experiment below, we report the non-interpolated average precision over the first 1000 rankings, measured in this way.

We used three pairs of relations from three different domains. In the business domain, we joined `Iontech` and `HooversWeb`, using company name as the primary key, and the string representing the "site" portion of the home page as a secondary key. In the movie domain, we joined `Review` and `MovieLink`, using film names as a primary key. As a secondary key, we used a special key constructed by a hand-coded normalization procedure for film names that is used in IM, an implemented heterogeneous knowledge integration system [Levy *et al.*, 1996a]. In the animal domain, we joined `Animal1` and `Animal2`, using common names as the primary key, and scientific names with a hand-coded domain-specific matching procedure as a secondary key.

The results are summarized in Table 2. On these domains, similarity joins are extremely accurate, relative to matching on the secondary keys. In the movie domain, the performance is identical to the hand-coded normalization procedure.

In all the domains, the matching procedure for the secondary keys is somewhat error prone, so these measurements are approximate. Checkings the pairings manually would be better, but far more expensive. To get some idea of the accuracy of the secondary keys we took the top 100 scoring pairs in the business domain, and manually checked the 13 pairs marked as "incorrect" according to the secondary key. Of these 13 pairings, there were 11 in which the secondary keys were wrong, one in which the pairing was wrong (at rank 77), and one pair where correctness could not be easily determined.

We also performed some other experiments using this data. In practise, one onerous task in writing knowledge integration tools is extracting the relevant information from a Web site; for instance, it may be difficult to extract the movie name from a movie review, if the format of reviews is sufficiently varied. In our model, this task corresponds to segmenting the tuples of each relation into fields.

However, similarity joins can still be performed even if tuples are not segmented. For instance, instead of extracting the movie name from a review, and representing the name as a second field in a relation, one could treat the entire review as a single field; similarly, one could treat an entire movie listing from `Movielink` as a single unsegmented document including a movie name, cinema address, and show times. Similarity joins can still be performed on the resulting fields, and documents referring to the same movie should still match reasonably well, although one would expect the irrelevant "noise" words to have some adverse affect. In our experiments with the `Review` and `Movielink` relations, however, this affect was quite: joining movie names to unsegmented movie reviews reduces average precision by only 2%, and joining unsegmented `Movielink` tuples to unsegmented reviews reduces average precision by less than 7%.

The average precision for the similarity joins are all extremely high, relative to standard IR benchmarks such as those used in the TREC conferences. This may be because, in all of these domains, there is a large pool of "right" answers—*i.e.*, there are many correct pairings from which to choose—and hence the retrieval algorithm can restrict itself, even in the top 1000 answers, to answers for which its confidence is fairly high. In a final set of experiments, we looked at *constrained similarity joins*—queries of the form

$$\texttt{p}(X_1,\ldots,X_i,\ldots,X_a) \ \wedge\ \texttt{q}(Y_1,\ldots,Y_j,\ldots,Y_b) \ \wedge\ X_i \sim Y_j \ \wedge\ \texttt{const}(D) \ \wedge\ X_k \sim D$$

In this case the similarity join is restricted to tuples for which some field is similar to D, where D is some document specified by the user (via the `const` relation). This is analogous to imposing an additional selection criterion on an ordinary relational join, and should produce a smaller pool of "correct" pairings.

These queries are somewhat difficult to evaluate automatically in general, so we were careful about choosing the additional "selectional criteria". In the movie domain, we required the review to be similar to one of two short plot descriptions, each of which were intended to pick out one particular movie.[8] An answer was judged correct if the pairing of film names matched the secondary key and involved the correct movie. In the business domain, we restricted the `industry` field of `HooverWeb` with several one- or two-word queries (*e.g.*, "modems" or "telecommunications equipment"). In the animal domain, we restricted the historical range with one-word queries (*e.g.*, "mexico".) In these domains an answer was judged correct if the restricted field contained all desired keywords and the pairing was correct (according to the secondary key).

[8] Plot1 was "depressing czech movie set in russia", for *Kolya*, and Plot2 was "jim carey comedy about a lawyer", for *Liar, Liar*.

The average precision for these WHIRL queries is lower, but still respectable—between 36% and 100%, and averaging around 56%. However, it is possible that the narrow interpretation of correctness used in these experiments is not always appropriate. We will illustrate this point with an anecedote. We evaluated the following WHIRL query

$\texttt{const1(I0)} \wedge \texttt{const2(A0)} \wedge \texttt{ReutersTest(Story,Keywords)} \wedge \texttt{Keywords}\sim \texttt{A0} \wedge \texttt{Story}\sim \texttt{I0}$

with `const1` containing the one-word document "restaurant" and `const2` containing the one-word document "acq". "Acq" is the controlled vocabulary code for stories about business acquisitions and mergers; thus this more-or-less traditional query asks for acquisition stories about the restaurant industry. It retrieves only two documents. The first of these begins "Management group has restaurant majority - Restaurant Associates Industries, Inc said the management group led by ..." and is clearly relevant. The second has a very low score, and mentions restaurants only in passing in a longish story on the petrochemical industry.

We also evaluated a WHIRL query which asks for restaurant-related acquisition stories using the `Hoovers` categorization of companies by industry:

$\texttt{const1(I0)} \wedge \texttt{const2(A0)} \wedge \texttt{ReutersTest(Story,Keywords)} \wedge \texttt{Keywords}\sim \texttt{A0}$
$\texttt{Hoovers(Company,Industry)} \wedge \texttt{Story}\sim \texttt{Company} \wedge \texttt{Industry}\sim \texttt{I0}$

with `const1` and `const2` as given above. The first story retrieved by the "traditional query" above appears seven times in the first 20 documents, even though "Restaurant Associates Industries, Inc" is *not* mentioned in `Hoovers`. Instead the story is *incorrectly* paired to several other restaurant organizations: "American Restaurant Group, Inc.", "Max & Erma's Restaurants, Inc", and five other restaurant chains.[9] Even more interestingly, of three other potentially relevant documents that appear in the top 10 pairings, two of these also result from incorrect pairings. The story for one of these pairs begins "Pepsico unit lifts stake - Taco Bell Corp, a unit of Pepsico Inc, said it increased its stake in Calny Inc...", and is paired with the company "Del Taco, Inc"; the second story is titled "Steinberg group has five pct of Pizza Inn", and is paired with "Domino's Pizza, Inc".[10]

In short, it is possible that regularities in entity names can cause even incorrect pairings to produce useful answer substutitions. This is an unexpected but intriguing result of our approach.

5 Conclusions

To summarize, we have outlined an approach to the integration of structured heterogeneous information sources, based on Datalog augmented with standard IR methods for reasoning about similarity. The approach is embodied in an implemented logic called WHIRL.

WHIRL is intended to address the integration of relations that are semantically heterogeneous in the sense that there is no common naming scheme for entities. This is a problem

[9]Of course, in the context of this query, it would be awkward to present all of these pairings to the user. It would be more appropriate to project out the bindings for `Story` and merge the different scores for it in some way.

[10]The third plausibly relevant acquisition story involves the International Multifoods Corp, which is described in Hoover as operating in the area of "Food - wholesale to restaurants".

that has not received much attention. Current knowledge 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.

WHIRL handles heterogeous data by allowing a query to join two relations by looking for pairs of tuples with similar, rather than identical, keys. The prior work in knowledge integration most similar to WHIRL is probably the work of Huffman and Steier on "heuristic joins" [1995]. However, they propose domain-specific heuristics for particular classes of entities, like person names and company names, and propose a much simpler and less general method for handling the uncertainty that results from heuristic joins.[11] Within the IR community, a number of DBMS systems have been proposed that combine ranked retrieval with database operations. However, we are not aware of any hybrid models that would, like WHIRL, efficiently support use of free text as keys; nor are we aware of any efforts to use such hybrid systems for knowledge integration problems.

Similarity in WHIRL is assessed using the vector-space model. Initial experiments show that the accuracy of WHIRL's "similarity joins" are quite good, even compared to hand-coded integration schemes based on normalization. WHIRL's theorem prover also makes use of indexing and optimization methods from information retrieval in searching for answer substitutions. As a consequence, the current implementation can handle join operations on moderate sized databases (containing a few tens of thousands of tuples) at interactive speeds. We note that many of the information sources used in current knowledge integration systems are of small or moderate size.

Although our early results are encouraging, many additional research issues remain to be addressed. WHIRL is not yet a complete method for knowledge integration, lacking methods for handling heterogeneous database schemata and other essential subtasks. There are many well-known methods for conducting an approximate $A*$ search; some or all of these may lead to further performance improvements. We also note that the current proposal 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], and learning logical expressions [Quinlan, 1990].

References

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

(Bartell *et al.*, 1994) Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew. Automatic combination of multiple ranked retrieval systems. In *Seventeenth Annual Interna-*

[11]Specifically, they propose thresholding each use of a domain-specific similarity metric at an appropriate level, rather than maintaining rankings based on a global estimate of similarity.

tional ACM SIGIR Conference on Research and Development in Information Retrieval, 1994.

(Boyan *et al.*, 1994) Justin Boyan, Dane Freitag, and Thorsten Joachims. A machine learning architecture for optimizing web search engines. Technical Report WS-96-05, American Association of Artificial Intelligence, 1994.

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

(Duschka and Genesereth, 1997a) Oliver M. Duschka and Michael R. Genesereth. 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 and Genesereth, 1997b) Oliver M. Duschka and Michael R. Genesereth. Query planning in infomaster. In *Proceedings of the Twelfth Annual ACM Symposium on Applied Computing (SAC97)*, San Jose, CA, February 1997.

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

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

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

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

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

(Levy *et al.*, 1996b) Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, august 1996.

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

(Nilsson, 1987) Nils Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufmann, 1987.

(Quinlan, 1990) J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3), 1990.

(Salton, 1989) Gerard Salton, editor. *Automatic Text Processing.* Addison Welsley, Reading, Massachusetts, 1989.

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

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