Keyword Join: Realizing Keyword Search for Information Integration

Bei YU[1] , Ling LIU[2] , **Beng Chin OOI**[1,3] and **Kian-Lee TAN**[1,3]
[1]Singapore-MIT Alliance, National University of Singapore
[2]College of Computing, Georgia Institute of Technology
[3]Dept. of Computer Science, National University of Singapore

***Abstract*— Information integration has been widely addressed over the last several decades. However, it is far from solved due to the complexity of resolving schema and data heterogeneities. In this paper, we propose out attempt to alleviate such difficulty by realizing keyword search functionality for integrating information from heterogeneous databases. Our solution does not require predefined global schema or any mappings between databases. Rather, it relies on an operator called *keyword join* to take a set of lists of partial answers from different data sources as input, and output a list of results that are joined by the tuples from input lists based on predefined similarity measures as integrated results. Our system allows source databases remain autonomous and the system to be dynamic and extensible. We have tested our system with real dataset and benchmark, which shows that our proposed method is practical and effective.**

***Index Terms*— keyword join, keyword query, data integration, database**

I. INTRODUCTION

Keyword search is traditionally considered as the standard technique to locate information in unstructured text files. In recent years, it has become a *de facto* practice for Internet users to issue queries based on keywords whenever they need to find useful information on the Web. This is exemplified by popular Web search engines such as Google and Yahoo.

Similarly, we see an increasing interest in providing keyword search mechanisms over structured databases [17], [18], [2], [10], [3]. This is partly due to the increasing popularity of keyword search as a search interface, and partly due to the need to shield users from using formal database query languages such as SQL or having to know the exact schemas to access data. However, most of keyword search mechanisms proposed so far are designed for centralized databases. To our knowledge, there is yet no reported work that supports keyword search in a distributed database system.

On the other hand, information integration has become a desired service for collecting and combining relevant information from multiple sources, due to the growing amount of digital information. In this paper, we present our solutions for facilitating keyword search in order to retrieve integrated textual information from heterogeneous databases. An obvious method to achieve such a goal is to add keyword search functionality over existing data integration systems [23], [11], [22], [9]. However, the traditional data integration approach requires systematic logical mappings to be built in prior, and it needs laborious manual work for schema matching. Therefore, it greatly constrains the applicability of data integration approach for large and dynamic data sources. Our approach is alternative to the traditional data integration approach, and we aim to apply our solutions in automatic, scalable and dynamic distributed database systems, such as P2P-based database systems.

The key idea of our approach is to join local answers generated by individual databases into a global answer with more complete information. Consider an illustrative example of two databases, DB1 and DB2, shown in Figure 1. Suppose an user issues a keyword query "Titanic, 1997, DVD". We can obtain the following local answers: from DB_1, we have one partial answer tuple t_1^1 = `(Titanic, 1997, Love Story, 6.9/10)` that contains the keywords "Titanic" and "1997"; and from DB_2, we have two partial answer tuples t_1^2 = `(Titanic, Paramount Studio, DVD, $22.49)` and t_2^2 = `(Titanic(A&E Document), Image Entertainment, DVD, $33.91)` both containing the keywords "Titanic" and "DVD". Now, we can join the two sets of local answers based on certain similarity criteria. For example, t_1^1 and t_1^2 can be combined based on their common column "`Titanic`" to get the final integrated answer `(Titanic, 1997, Love Story, 6.9/10, Titanic, Paramount Studio, DVD, $22.49)`. This combined tuple aggregates the data from the two databases can hopefully render some useful information to the user. Similarly, t_1^1 and t_2^2 can be combined. Thus, we can return two complete answers to the keyword query for the user to select.

We introduce the notion of *keyword join* for joining lists of local answer tuples to generate global answers based on similarities between columns, which are measured based on standard IR methods and heuristic-based string matching techniques. Consequently, our proposed solution does not require global level schema mappings or any mediated schema. Thus it is suitable for dynamic environment where both the data sources and the whole system keep changing.

As we can see, another different feature of our system from traditional data integration systems is that the results

Bei YU is with the Computer Science Program, Singapore-MIT Alliance. Tel: 68744774. E-mail: yubei@comp.nus.edu.sg.

Ling LIU is with the College of Computing, Georgia Institute of Technology. E-mail: lingliu@cc.gatech.edu.

Beng Chin OOI is with the Department of Computer Science, National University of Singapore. E-mail: ooibc@comp.nus.edu.sg.

Kian-Lee TAN is with the Department of Computer Science, National University of Singapore. E-mail: tankl@comp.nus.edu.sg.

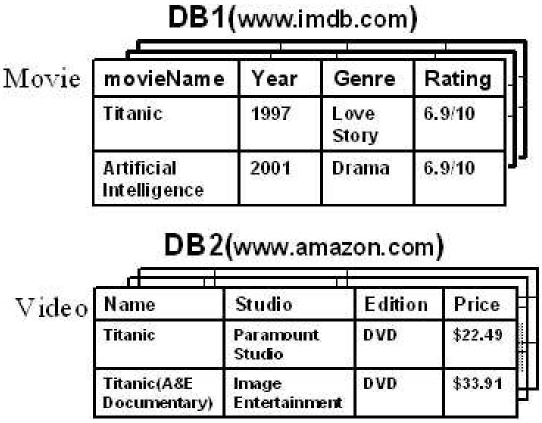


Fig. 1. An example.

returned by our system is a ranked list of potentially relevant answers, resulting from the fuzziness of keyword queries and the heuristic nature of keyword join. We believe this is an acceptable user interface since it is already widely adopted today.

We make the following contributions in this paper:

- We design an alternative data integration framework based on keyword search that can alleviate the difficulty of existing data integration systems.
- We define and implement a novel *keyword join* operator for integrating heterogeneous tuples from different databases and return top-K answers.
- We make extensive experiments of evaluate the feasibility and performance of our proposed solution.

The rest of the paper is structured as follows. Section II presents the general framework of our integration system. Section III describes our implementation of keyword join, which includes decision of similarity measure and a top-K processing algorithm. In Section IV, we evaluate the performance of our system with various datasets. Section V describes related work to our proposed solution, and finally Section VI concludes the paper and discusses our future work.

II. Framework

Generally, our proposed system consists of 3 components: the database selector, the keyword search engine for local database, and the local answer integrator. The database selector selects a subset of relevant databases given a keyword query. The keyword search engine for local database receives a keyword query and generates a list of joined tuples that have partial or complete query keywords. The local answer integrator, which is the focus of this paper, handles all the local answer lists from the relevant databases, selects suitable combinations of local answer lists, and joins them with keyword join operator to generate a ranked list of global answers — integrated tuples — to return to users. We will describe these 3 components sequentially in the following subsections.

A. Keyword query model & database selection

A keyword query is a list of *keywords combinations*, and a keyword combination consists of one or more keywords. For example, "programming language, compiler" is a two-keyword-combination query, and it contains 3 keywords: "programming", "language", and "compiler". Introducing the concept of keyword combination into our keyword query model is due to the need to guarantee the proximity between some of the keywords when aggregating local answers from different databases — keywords of a keywords combination must be present together in at least one database in any final integrated result.

The source databases are assumed to be text-rich. The terms in the databases, including both the tables and the metadata, are indexed with an inverted index, i.e., each term is mapped to the database that contains it. The inverted index is used to select a subset of relevant databases for a keyword query.

B. Local keyword query processing

We apply the keyword search engine described in [17] for generating local answers at each selected database.

A keyword index is necessary for performing keyword search over a database [17]. It is an inverted index that associates each appearance of keywords in the relations with a list of its occurrences. Most commercial DBMSs support such an index. In our implementation, we use MySQL, which has *fulltext* indexing and search functionality adequate for keyword search, as the DBMS for each database.

When a database receives a keyword query Q, it first creates a set of *tuple sets* using its local keyword index. Each tuple set R^Q is a relation extracted from one relation R of the database, containing tuples having keywords of Q, i.e., $R^Q = \{t | t \in R \wedge Score(t, Q) > 0\}$, where $Score(t, Q)$ is the measure of the relevance of tuple t in relation R with respect to Q, which is computed with the local index according to standard IR definition [17]. Next, with these tuple sets, the system generates a set of Candidate Networks (CNs), which are join expressions based on foreign key relationship between relations, capable of creating potential answers. By evaluating these generated CNs, the database can finally produce its local answers — trees of joining tuples from various relations. Each tuple tree T is associated with a score indicating its degree of relevance to the query, which is calculated as

$$Score(T, Q) = \frac{\sum_{t \in T} Score(t, Q)}{size(T)}, \quad (1)$$

where $size(T)$ is the number of tuples in T.

Recall the example of Figure 1, the scores of the partial answers given by DB1 and DB2 are: $score(t_1^1) = 2.31; score(t_1^2) = 0.86, score(t_2^2) = 0.78$.

Any two tuple trees are said to be *heterogeneous* if they have different schemas. The tuple trees generated from different CNs are heterogeneous, so are the tuple trees generated by different databases.

The number of answers generated by each database is limited by a system parameter k_l, which could be adjusted to the degree of relevance of the databases to the query.

C. Integration of local answers

The final results for a keyword query are a ranked list of integrated local answers based on some similarity measure, which will be described in Section III-A.

DEFINITION 1 *A local answer to a keyword query is a tuple generated by an individual database that contains at least one keyword combination in the query.*

Local answers with complete keyword combinations in the query are called *local complete answers*; otherwise, they are called *local partial answers*.

DEFINITION 2 *A global answer to a keyword query is a joined network of local answers from different databases, and it has all the keywords in the query.*

The operation to combine local answers into global answers is called *keyword join*. It is a similarity-based join operator, and it also considers the appearance keyword combinations in tuples.

DEFINITION 3 *Given a keyword query Q, a set of lists of local answers ($L_1, L_2, \cdots, L_p$), together with a threshold O_T, the keyword join $L_1 \bowtie_k L_2 \bowtie_k \cdots \bowtie_k L_p$ returns all set of integrated tuples ($t_1, t_2, \cdots, t_p$) such that (1) $t_1 \in L_1$, $t_2 \in L_2$, $\cdots$, $t_p \in L_p$, (2) ($t_1, t_2, \cdots, t_p$) has all the keywords in the query, and (3) $t_1, t_2, \cdots, t_p$ are connected into a network of joined tuples such that for each adjacent pair of tuple t_i and t_j, $overlap(t_i, t_j) \geq O_T$.*

Note that the input of keyword join is a set of *lists* instead of tables. This is because the input tuples are heterogeneous: they are generated from different CNs, or from different databases. Consequently, the joinablility between two heterogeneous tuples are determined by their information overlapping — $overlap(t_i, t_j)$, which is determined by the existence of similar or matching columns between two tuples. In addition, different from ordinary join operation, keyword join is non-associative.

THEOREM 1 *Keyword join is not associative, i.e.,*

$$(L_1 \bowtie_k L_2) \bowtie_k L_3 \neq L_1 \bowtie_k (L_2 \bowtie_k L_3).$$

Proof: Suppose $t_1 \in L_1$, $t_2 \in L_2$ and $t_3 \in L_3$, and integrated tuple $t_2 - t_1 - t_3 \in (L_1 \bowtie_k L_2) \bowtie_k L_3$. According to Definition 3, $t_2 - t_1 - t_3$ cannot be generated by $L_1 \bowtie_k (L_2 \bowtie_k L_3)$. Thus keyword join is not associative. ∎

Each global answer $T = (t_1, t_2, \cdots, t_n)$ is associated with a score to measure its relevance to the query. It is defined as

$$score(T) = \frac{\sum_{t \in T} score(t)}{size(T)},$$

where $size(T)$ is the number of local answers T aggregates. This is based on the intuition that a global answer integrated from local answers with higher scores would be more relevant to the query; meanwhile, since our integration is based on heuristic measure, global answer with more sources would incur more "noise", and then less relevant.

The global answers generated by keyword join operator fall into 3 cases: (1) the global answers are joined by local complete answers only; (2) the global answers are joined by local partial answers only; and (3) the global answers are joined by both local partial answers and local complete answers.

III. IMPLEMENTATION OF KEYWORD JOIN

Having understand the definition of keyword join, we now discuss its implementation details in this section. Subsection III-A describes the similarity measure to determine the portion of overlapping of two tuples. Then in subsection III-B, we present a top-K processing algorithm for evaluating keyword join efficiently.

A. Similarity measure

Suppose we have two tuples t_1 and t_2, and they have l_1 and l_2 columns respectively: $t_1 = (c_1^1, c_2^1, \cdots, c_{l_1}^1)$, and $t_2 = (c_1^2, c_2^2, \cdots, c_{l_2}^2)$. The purpose of the similarity measure is to determine the eligibility of combining t_1 and t_2 to render meaningful information, by assigning an overlap score for the pair, denoted as $overlap(t_1, t_2)$.

The general idea is pair-wise comparing the columns between t_1 and t_2. The similarity score between columns contributes to the overlap score between the two tuples. In particular, there are three specific steps, which are described sequentially in the following.

1) Selection of significant columns: It is obvious that it would be neither computationally efficient nor semantically appropriate to compare every pair of columns between t_1 and t_2. In the example of Figure 1, we should not compare the column value "6.9/10" in DB1 with the column value "$22.48" in DB2. Therefore, we introduce the concept of *significant column*, and only every pair of significant columns from t_1 and t_2 are compared.

In order to identify significant columns from a tuple effectively, we introduce the concept of the *distinctness* of column values with an attribute A in a table T. Suppose the total number of column values with attribute A in T is $S(T, A)$, and the total distinct number of column values with attribute A is $V(T, A)$. The distinctness of the values of A, denoted as $D(A)$, in T is

$$D(T, A) = \frac{V(T, A)}{S(T, A)}.$$

Distinctness $D(T, A)$ measures the importance of column values of attribute A in a table instance T. Obviously, the distinctness of primary key of a table is always 1. Observe the tuples of the Movie table in $DB1$ in Figure 1, $D(Movie, movieName) = 1$, and $D(Movie, Rating) = 0.5$. It means that the column values of *movieName* is more important than that of *Rating* for identifying tuples in the table.

DEFINITION 4 *Given a distinctness threshold D_T, a column c in a tuple t and table T is a* significant column *if (1) $D(c, T) > D_T$, and (2) c comprises textual content.*

Intuitively, a significant column is sufficient to identify the tuple from the table parameterized by D_T (with respect to the distinctness threshold), and it is self-describing. Refer to the example in Figure 1, column values such as "1997" and "6.9/10" are not significant columns since they are numerical values — their semantic meaning can only be understood together with the table schema. The column value "DVD" is textual and its meaning can be inferred (although there could be ambiguities, it is an unavoidable problem in text processing), but it should not considered as significant column too, as it is not a distinctive column value in the table.

In implementation, the distinctness value of each column, and a boolean value indicating whether a column is full-text indexed or not are tagged in the local answer generated by each database. Therefore, it is easy to identify the significance of a column when performing keyword join.

Our definition of significant column described above is for automatically detecting significant columns from a tuple. It is no doubt that a system administrator or an advanced user can decide *significant* columns more effectively. However, it is usually neither feasible nor efficient to involve human invention in a large scale and dynamic system.

In our experiments, we will show the comparison of the results when the choice of significant columns varies.

2) Pair-wise textual column comparison: Given a pair of significant columns c_1 and c_2, we need to assign a score to indicate their similarity. There are many ways to evaluate the similarity between textual fields in literature [13], [7], [5]. What we apply in our solution is the TF.IDF method which is effectively used in the WHIRL system [5].

Each column c_i with attribute A from table T has a set of terms $c_i = t_1^i, t_2^i, \cdots, t_{n_i}^i$ (after removing stop words). We measure the TF (term frequency) and IDF (inverse document frequency) values for each term. The calculation of TF and IDF values is similar to what defined in text document retrieval []. TF of a term t_j^i is the number of its occurrences in c_i. IDF of t_j^i measures the fraction of the total number of tuples in T to the total number of tuples in T that contains it in their attribute A. The weight of a term is determined by its TF and IDF value: $w = TF \cdot IDF$. The similarity score between two columns c_i and c_j is

$$sim(c_i, c_j) = \frac{\sum_{t \in C(c^i, c^j)} w_t^i \cdot w_t^j}{\sqrt{\sum_{t \in c^i} w_t^2 \cdot \sum_{t \in c^j} w_t^2}},$$

where $C(c_i, c_j)$ is the set of common tokens between c_i and c_j.

In our proposed system, the weight of the terms in a tuple is provided locally by each source database along with its local answers.

3) Developing overlap score: The score for measuring the overlap portion between two tuples is derived from the individual scores of the significant column pairs between them. Considering that the most matching column pairs should be the linkage between the tuples, we set their overlap score as the maximal similarity score among all their significant column pairs, i.e.,

$$overlap(t_1, t_2) = \max_{1 < i \leq l_1, 1 < j \leq l_2} sim(c_i^1, c_j^2),$$

where c_i^1, c_j^2 are significant columns from t_1, t_2 respectively. Thus, by setting an appropriate overlap threshold, we can determine if two tuples are joinable by examining their overlap score.

B. Top-K processing for keyword join

We address the problem of how to generate top K results efficiently when performing keyword join to a set of partial answer lists in this section. Note that when we perform keyword join operation on multiple input lists, it is difficult to use traditional query evaluation plans, such as left deep tree, right deep tree, etc., since keyword join is not associative. We can only join a lists of partial answers by examining every combination of tuples extracted from the input lists respectively. Figure 2 shows the algorithm to integrate a combination of tuples $t_1, t_2, \cdots, t_p$ given query Q and overlap threshold O_T.

```
join(Q, O_T, t_1, t_2, ..., t_p)
    // checking the keywords in the combination of tuples
    if the union of the sets of keywords of t_1, t_2, ..., t_p is a
    subset of Q
        return null
    Create two empty lists L_connected, and L_unCompared
    // checking the "connectivity" of t_1, t_2, ..., t_p
    Put t_1 into L_connected, put all the others into L_unCompared
    while L_unCompared is not empty
        if there are two tuple trees t and t' from L_unCompared
        and L_connected respectively, such that
        overlap(t, t') >= O_T
            Remove t from L_unCompared
            Put t into L_connected
            Put t into adj(t') // adjacent list of t'
        else
            return null
    Combine the tuples in L_connected into an integrated
    tuple T
    return T
```

Fig. 2. Join a combination of tuples.

To perform top-K processing for keyword join efficiently, we employ the ripple join [14], which is a family of join algorithms for online processing of multi-table integration queries. In the simplest version of the two-table ripple join, one tuple is retrieved from each table at each step, and the new tuples are joined with all the previously-seen tuples and with each other. This process is essentially a spanning in the Cartesian product space of input tables, where each table corresponds to a dimension, and getting valid join results.

In our context, the input tables are lists of ordered tuples, and the combination score of the joined result is calculated based on the monotonic Equation II-C. Therefore, in the spanning space during joining a set of tuple lists, the score of the combination of tuples at each point is less than that of the combinations of tuples previously seen along each dimension. The small arrows in Figure 3 indicate the decreasing sequence of the scores of the combinations of tuples. This property enables us to prune the spanning space for generating top K join results efficiently.

The pruning process works as follows. In each step, when we retrieve one tuple from each input list, we join each new tuple with previously-seen tuples in all other lists, in descending order of their relevance scores. In other words, the examination of the combinations of tuples is towards the decreasing direction along each dimension. For example,

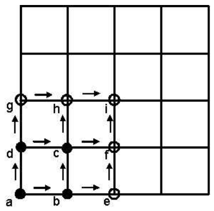


Fig. 3. An example of pruning spanning space.

in Figure 3, which is at step 3 of ripple join between two input lists, the next sequence of combinations of tuples for examination would be $< e, f, g, h, i >$. Therefore, before we examine the validity of each combination of tuples at a point, we first calculate its combination score. At the same time, a list L_{ans} is used to store the top K join results we currently have. We then compare the combination score with the K-th largest score in L_{ans}, and if the former is smaller, we can prune it and all the rest points along that dimension. For example, as in Figure 3, suppose we are going to examine the validity of point g. We first calculate its combination score, if the score is smaller than the current K-th largest score in L_{ans}, we can safely prune the remaining points along that dimension, i.e., points $< h, i >$, since their scores must be smaller than that of point g.

In addition, if in a step all the points along all dimensions are pruned — meaning that the points in the rest of the space that have not been spanned all have smaller scores than the current K-th largest score — the algorithm could be stopped. For instance, in Figure 3, if the scores of points e and g are both smaller than the current K-th largest score, all the points in this step are pruned, and consequently we can stop the algorithm and return the current top K results.

$keywordJoin(K, Q, O_T, L_1, L_2, \cdots, L_p)$
1. Set p pointers $pt_1, pt_2, \cdots, pt_p$, pointing to the top unscanned tuples of $L_1, L_2, \cdots, L_p$, respectively
2. Set S_{low} as the K-th lowest score of the joined results obtained so far
3. **while** there is unscanned tuple in $L_1, L_2, \cdots, L_p$
4. Set boolean variable $allPruned \leftarrow$ **true**
5. **for** $i \leftarrow 1$ to p
6. Get next tuple $L_i[pt_i]$ from L_i
7. **if** $score(L_1[1], \cdots, L_i[pt_i], \cdots, L_p[1]) \leq S_{low}$
// all points along i dimension are pruned
8. **go to** 5
9. $allPruned \leftarrow$ **false**
10. Set variables $id_1, \cdots, id_{i-1}, id_{i+1}, \cdots, id_p$ to 1
11. **for** $k \leftarrow 1$ to p and $k \neq i$
12. **for** $id_j \leftarrow 1$ to $pt_j - 1$ and $j \leftarrow 1$ to p and $j \neq i, k$
13. **for** $id_k \leftarrow 1$ to $pt_k - 1$
14. **if** $score((L_1[id_1], \cdots, L_k[id_k], \cdots, L_i[pt_i], \cdots, L_p[id_p])) \leq S_{low}$
// rest points are pruned
15. **go to** 12
16. $IT = join(Q, O_T, L_1[id_1], \cdots, L_k[id_k], \cdots, L_i[pt_i], \cdots, L_p[id_p])$
17. **if** $IT \neq$ **null**
18. Put IT into L_{ans}
19. Update S_{low}
20. Increase pt_i
21. **if** $allPruned =$ **true**
22. **return** L_{ans}
23. **return** L_{ans}

Fig. 4. Keyword join algorithm.

The above pruning process can be easily extended to the keyword join on multiple input lists. Figure 4 shows the keyword join algorithm to produce top K integrated tuples from a set of lists $L_1, L_2, \cdots, L_p$.

Data Set	# databases	# tables	# tuples
movies-actors	2	2	18452
amalgam	4	56	107612
thalia	28	64	2402

TABLE I
DATA SETS STATISTICS.

IV. EXPERIMENTAL RESULTS

A. Datasets and queries

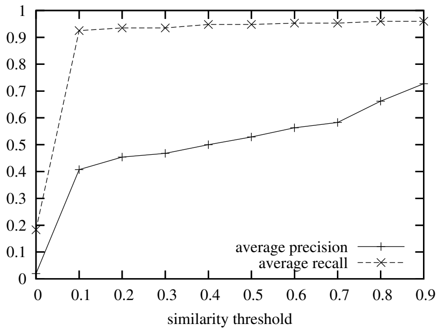
We use 3 datasets in different domains to test the quality of the returned results of our keyword join operator. The first dataset is movies-actors dataset containing two databases — movies and actors. It is downloaded from Niagara project page [1]. The second dataset is amalgam data integration test suite obtained from [24]. It includes 4 databases in computer science bibliography domain, which are developed by 4 separate students. The third dataset is thalia data integration benchmark [16]. It consists course catalog information from computer science departments around the world. The data sources are originally stored in XML format, we converted 28 sources from the testbed into relational format with Shrex [8]. Table I summarizes the statistics of the 3 datasets.

Among the 3 datasets, only thalia testbed provide 12 benchmark queries. We have to generate queries by ourselves for the other two. For the movies-actors dataset, we generate 100 queries which are actor names and director names that have worked for the same movie. Some sample queries are: "Bruce Willis, Renny Harlin", "Keanu Reeves, Francis Coppola", etc.. For the amalgam test suite, we find out 139 authors that co-exist in more than one databases, and use their names as keyword queries. The relevance data for the queries are generated by evaluating SQL queries to a temporary database that have tables from different databases in dataset.

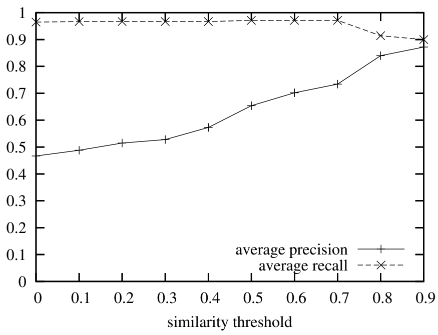
In the following sections, we first valuate the effect of different system parameters to the quality of results generated by keyword join movies-actors and amalgam dataset. The parameters that we vary in the experiment are: overlap threshold (Ot), distinctness threshold (Dt), number of retrieved local answers per database (Ln), and number of retrieved global answers (Gn). Then we demonstrate the integration capability of our solution with the 12 benchmark queries of thalia testbed.

B. Evaluation of system parameters

1) Effects of the overlap threshold: Figure 5 shows the variance of precision and recall when the overlap threshold increases. The results are similar as what we expected. When the overlap threshold is higher, the precision of the results becomes better because more irrelevant results are filtered. Observe that the recall keeps unchanged when overlap threshold varies in a large range for both datasets. This shows that correct answers are returned together with irrelevant answers when overlap threshold gets lower. For the amalgam dataset, when overlap threshold exceeds 0.7, recall begins to drop. This is because that some good results are also filtered due to high threshold. Movies-actors dataset does not have this phenomena because the correctness of the results are generally based on exact match of movie titles — correct results must have high overlap score — so good results can only be filtered when threshold is extremely high. Note that when overlap threshold equals 0, amalgam dataset also shows relative high precision and recall. We find out two reasons. First, the columns that match the keywords are the same as the column that needs to be compared for two tuples, so most joined results have very high overlap score, and have high relevance to the query. Second, the returned results of keyword join for this dataset is very few for most of the queries, that is, good results are also returned when threshold is low.



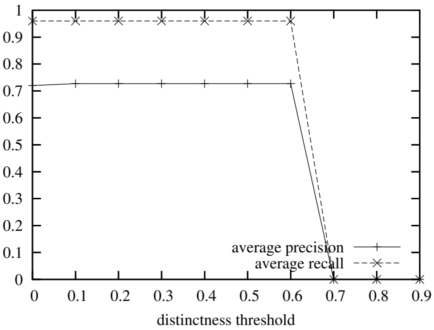
(a) movies-actors dataset



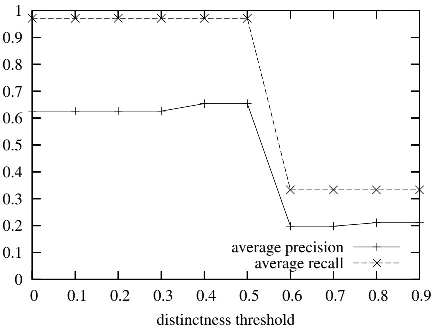
(b) amalgam dataset

Fig. 5. Effect of the overlap threshold (Dt = 0.5, Gn = 10, Ln = 250).

2) Effects of the distinctness threshold: Figure 6 presents the precision and recall when the distinctness threshold changes, which affect the selection of significant columns. According to the figures, the precision and recall is slightly lower when distinctness threshold is very small. This shows that more columns are identified as significant columns when distinctness threshold is low, and comparisons between more column pairs leads to more noise in the result. Also note that when distinctness threshold is too high, the precision and recall drop a lot. It is because most "real" significant columns are missed and relevant results cannot be generated. Note that for the movies-actors dataset, precision and recall becomes zero when distinctness threshold exceeds 0.7, which shows that no "real" significant columns is selected at all.



(a) movies-actors dataset

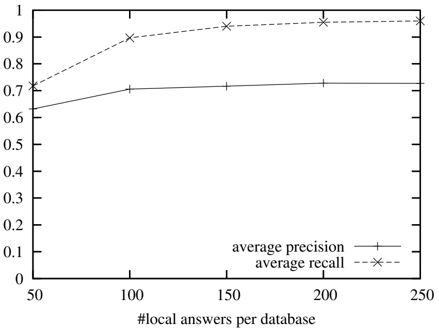


(b) amalgam dataset

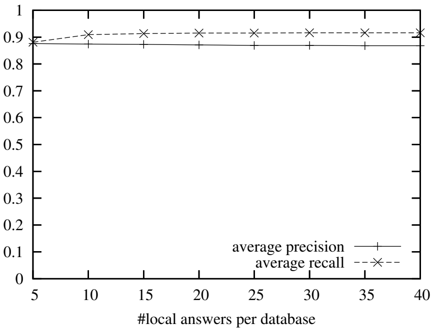
Fig. 6. Effect of the distinctness threshold (Ot = 0.9, Gn = 10, Ln = 250).

3) Effects of the number of local answers: Figure 7 illustrates the changes of precision and recall when the number of local answers retrieved from each database increases. For both datasets, recall first becomes larger when the number of local answers increases, and finally reaches a stable value. This is reasonable because initially when number of local answers gets larger, additional relevant partial answers are included, and the recall increases. When the number of local answers is so large that no more relevant partial answers is added, the recall will keep the same. On the other hand, observe that precision increases slowly or even decreases slightly when the number of local answers increases until a stable value is reached. We think the reason is that more irrelevant local answers are put into input lists when number of local answer increases (remember that local answers are ranked and only top answers are retrieved), which will affect the keyword join results.

4) Effects of the number of global answers: Figure 8 shows the changes of precision and recall when the required number of global answers varies. Observe that recall increases when the number of global answers increases. Obviously the reason is that more good results are collected when the number of global answers gets larger. When the required number of global answers becomes too large, and there is no more actual global answers can be really generated, the recall will keep the same. Actually, when we examine the generated results in this experiment, we find that for most queries, the maximum number of available global answers is few, usually below 10. Precision does not show much variance when the number of required global answers increases. We think this is due to the reason stated above too, otherwise, precision should decreases

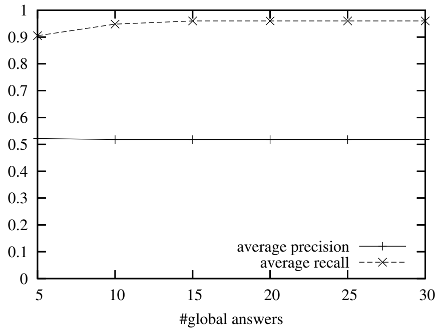


(a) movies-actors dataset

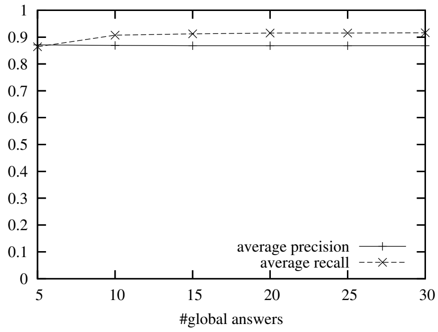


(b) amalgam dataset

Fig. 7. Effect of the number of local answers per database (Dt = 0.5, Gn = 10, Ot = 0.9).



(a) movies-actors dataset



(b) amalgam dataset

Fig. 8. Effect of the number of retrieved global answers (Dt = 0.5, Ot = 0.5, Ln = 50).

when the number of global answers increases.

C. Integration capability

Thalia dataset is associated with 12 benchmark queries[1] that represent different structural and semantic heterogeneities cases [16]. The original benchmark queries are in XML format. We transformed them into keyword queries by simply extracting the the phrases in the query that refers to objects in the dataset. For example, for the query: List the title and time for computer network courses.

```
 FOR $b in doc('cmu.xml')/cmu/Course
WHERE $b/CourseTitle ='%Computer Networks%'
RETURN <Course>
<Title>$b/Title</Title>
<Day>$b/Day</Day>
<Time>$b/Time</Time>
</Course>
```

We transform it into keyword query: “computer networks”. Our experimental results with the benchmark is as follows.

- Query 1 (renaming columns): the query can be successfully answered by returning the correct result in top global answers.
- Query 2 (24 hour clock): this needs conversion between different time representations. Our system cannot support it currently.
- Query 3 (union data types): the query can be successfully answered by returning the correct result in top global answers.

[1]The queries can be browsed at http://www.cise.ufl.edu/research/dbintegrate/packages/queries.xml

- Query 4 (meaning of credits): needs conversion between different representation of course units. It is difficult to realize it with our system.
- Query 5 (language translation): needs translation between different languages. It is difficult to realize it in our current system.
- Query 6 (nulls): the query can be successfully answered by returning the correct result in top global answers.
- Query 7 (virtual attributes): it involves semantic translation. Our system cannot support it.
- Query 8 (semantic incompatibility): same as Query 7.
- Query 9 (attribute in different places): our system cannot support it directly. But if provided with the information that “room” information is stored in the column with “time” attribute, i.e., put “time” in the query keywords, our system can easily answer the query.
- Query 10 (sets): same as Query 9.
- Query 11 (name does not define semantics): Our system can answer it by returning correct answers in the top integrated results.
- Query 12 (run on columns): Our system can answer it by returning correct answers in the top integrated results.

To conclude, our system could deal with 5 queries easily, and another 2 queries with small amount of metadata, which we think should not be a problem. Compared with the experimental results of the other two integration systems, Cohera and IWIZ, reported in [16], our system can solve less queries than them (both of them could do 9 queries with varying amounts of user-defined code.). However, our system does not require any costume code, and it is much easier to exploit.

V. Related Work

In this section, we shall present related work in keyword search and data integration systems.

A. *Keyword search in centralized databases*

Keyword search over centralized databases has been recently studied by several works. The most representative ones include the DISCOVER project [17], [18], the DBXplorer project [2], the BANKS project [10], and the ObjectRank work [3]. Given a set of query keywords, the query processing in DISCOVER finds the corresponding *tuple sets*, which are essentially relations that contain one or more keywords. It then exploits the schema of the database and generates Candidate Networks (CNs) that are join expressions for producing potential answers. DBXplorer [2] shares a similar design as DISCOVER. The BANKS system [10] models a relational database as a graph in which tuples are the nodes and the edges represent foreign-key relationships. Query answering essentially involves finding the subgraph that connects nodes matching the keywords, using heuristics during the search. ObjectRank [3] adopts a different approach: It extends the PageRank techniques for ranking web pages to rank the relevance of objects in databases, where the database is modeled as a labeled graph.

B. *Data integration systems*

Most of the data integration systems developed so far, such as TSIMMIS [9], Information Manifold [22], COIN [12], etc, require designing a global schema and the necessary mappings between the global schema and the source schemas. These steps are usually labor-intensive, needing manual intervention, and can only be performed offline.

Recently, there are some P2P data management systems proposed that do not require a centralized global schema [4], [15], [26]. They typically define mappings in the system to associate information between different peers. Queries could be posed to any peer, and the peer evaluates the query by exploiting the mappings in the system. These P2P systems differ from one another in terms of the concrete formalism used to define mappings between the peer schemas. [4] introduces the Local Relational Model (LRM) as a data model specifically designed for P2P data integration systems. In LRM, the interaction between peer databases is defined with coordination rules and translation rules. [15], [26] propose a peer data management system (PDMS) to manage structured data in a decentralized manner. It describes a formalism, named PPL (Peer-Programming Language), for defining mappings between peer schemas. [26] provides the query reformulation algorithm that reformulates a query Q over a peer schema to a query Q' over the actual data sources in the peers.

C. *Integration based on textual similarity*

WHIRL [5], [6] is a logic for database integration, which incorporates reasoning about the similarity of pairs of names from different sources, as well as representations of the structured data like conventional DBMS. The similarity is WHIRL is measured based on the standard TF.IDF method from information retrieval literature. Answers to a WHIRL query is a set of "best" tuples that have highest similarities. Text Join [13], based on the same semantics as WHIRL [5], devises techniques for performing text joins efficiently in a unmodified RDBMS. Although our proposed keyword join is also based on statistical measures of similarity between texts, our integration approach greatly differs from WHIRL in that we do not need any predefined and fixed schema mappings between data sources. In addition, WHIRL considers joining tuples from two input tables with fixed schemas, while our keyword join operates on multiple input lists of heterogeneous tuples.

D. *Database sharing without integration*

There are also some works that provide database sharing in P2P networks without relying on pre-defined schema mappings, such as PeerDB [25], PIER [19], and the mapping table approach [21], [20]. PIER [19] describes a query processing method that is intended to be scalable over the Internet. But the database schemas in PIER are assumed to be unique over all the peers. PeerDB [25] and the mapping table approach [20] share similar concept in that they both achieve data sharing between different peers by translating queries of one peer to the queries of another. The difference is that PeerDB uses IR technique to translate the query, while [21], [20] utilizes mapping tables [21]. However, neither of these works provides information integration capability such as join operations among peer databases. In contrast, the keyword join system proposed in this paper aims at achieving both information sharing and information integration among the network of databases shared by the peers.

VI. Conclusion

We have presented a framework for realizing keyword search for integrating information from heterogeneous databases. Our proposed system avoids complex data integration, making it suitable for dynamic and ad-hoc environments and cost effective in terms of implementation. We have also proposed an efficient algorithm for generating top K global answers with our proposed keyword join operator.

References

[1] NIAGARA Experimental Data. http://www.cs.wisc.edu/niagara/data.html.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE*, 2002.

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-Based Keyword Search in Databases. In *VLDB*, 2004.

[4] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *Workshop on the Web and Databases*, 2002.

[5] William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD*, 1998.

[6] William W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Trans. Inf. Syst.*, 18(3):288–321, 2000.

[7] William W. Cohen, Pradeep Ravikumar, and Stephen Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *IIWeb*, 2003.

[8] Fang Du, Sihem Amer-Yahia, and Juliana Freire. Shrex: Managing xml documents in relational databases. In *VLDB*, 2004.

[9] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117 – 132, 1997.

[10] G.Bhalotia, A.Hulgeri, C.Nakhe, S.Chakrabarti, and S.Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *ICDE*, 2002.

[11] C. H. Goh. *Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems*. PhD thesis, Massachusetts Institute of Technology, 1997.

[12] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel. Context interchange: new features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems (TOIS)*, 17(3):270 – 293, 1999.

[13] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text Joins in an RDBMS for Web Data Integration. In *WWW*, 2003.

[14] P. Haas and J. Hellerstein. Ripple Joins for Online Aggregation. In *SIGMOD*, 1999.

[15] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *ICDE*, 2003.

[16] Joachim Hammer, Michael Stonebraker, and Oguzhan Topsakal. THALIA: Test Harness for the Assessment of Legacy Information Integration Approaches. In *ICDE*, 2005. Poster Paper.

[17] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In *VLDB*, 2003.

[18] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *VLDB*, 2002.

[19] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *VLDB*, 2003.

[20] A. Kementsietsidis and M. Arenas. Data Sharing Through Query Translation in Autonomous Sources. In *VLDB*, 2004.

[21] A. Kementsietsidis, M. Arenas, and R. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *SIGMOD*, 2003.

[22] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Information Gathering from Heterogeneous, Distributed Environments*, AAAI Spring Symposium Series, 1995.

[23] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, 2002.

[24] Renée J. Miller, Daniel Fisla, Mary Huang, David Kymlicka, Fei Ku, and Vivian Lee. The Amalgam Schema and Data Integration Test Suite. www.cs.toronto.edu/ miller/amalgam, 2001.

[25] W. S. Ng, B. C. Ooi, K. L. Tan, and A. Y. Zhou. PeerDB: A P2P-based system for distributed data sharing. In *ICDE*, 2003.

[26] I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer-Data Management Systems. In *SIGMOD*, 2004.