From: Coward, Jeffery

Sent: 10/31/2013 12:34:26 PM

To: TTAB EFiling

CC:

Subject: U.S. TRADEMARK APPLICATION NO. 85561168 - DEEP WEB INTELLIGENCE - 4335.14US01 - Request for Reconsideration Denied - Return to TTAB - Message 4 of 5

****

Attachment Information:

Count: 5

Files: RDDW2-11.jpg, RDDW2-12.jpg, RDDW2-13.jpg, RDDW2-14.jpg, RDDW2-15.jpg

http://webcache.googleusercontent.com/search?q=cache:WRDbUx-VNV4J:ftp://ftp.cse.ohio-state.edu/pub/tech-report/2008/TR10.pdf+&cd=7&hl=en&ct=clnk&gl=us 10/29/2013 01:27:41 PM

type relations.

4.2 Data Source Representatives

We introduce a new concept, representative of a data source. Each data source schema has a list of output attributes which characterize the main focus or interest of the data source. If we know the main focus of a data source, we can use this information to rank the data source's relevance with respect to a keyword. For example, data source A has 20 output attributes. We find that 15 output attributes come from the concept term SNP, and the rest 5 comes from the concept term Protein. With this information, we roughly know that data source A is about SNP and protein data, and the main focus of A is providing SNP information. Data source B has 10 output attributes, and we find that all the 10 terms come from the Protein node. Data source B is built solely for providing protein information. Given the above information, if a user keyword can be mapped to the protein node in the ontology graph, we can have a high confidence in concluding that the data source B can provide more relevant information than A.

We define the representatives of a data source to be all biological concept terms in the ontology which can be reached by reverse traversing F or C type links starting from the data source's output attribute. Suppose a data source D has n representatives $r_1, r_2, ..., r_n$, and each representative r_i is associated with a weight w_i. The weight w_i is the ratio

Suppose a query contains a set of keywords $Q = \{k_1, ..., k_m\}$. A node corresponds to a data source schema. Given a node n_i, we first define the node score of n_i with respect to query keyword k_j as $NScore(n_i, k_j) = c_{ij} \times q_{ij} \times DSN_k$. Here, c_{ij} is the node coverage score of the node n_i with respect to k_j, and q_{ij} is the node quality score of node n_i with respect to k_j. If multiple data source can provide this keyword k_j, i.e. DSN_k is small, the score of this node n_i is decreased.

Node Coverage Score: The node coverage score of node n_i with respect to k_j is defined as $c_{ij} = \frac{1}{Level(i)+1} \times$ Is Contain(j). Level(i) is the shortest distance in terms of the number of edges from any of the starting points to the current node n_i. Is Contain(j) is a function which returns 0 if data source n_i does not contain k_j as its output, returns 1 otherwise. Note that the Level(i) value can vary across queries, because different queries have different starting points. This ranking function gives a node higher score if it covers more keywords and is easier to reach.

Node Quality Score: The node quality score of a node n_i with respect to k_j is defined as q_{ij} = OntoScore(i, j). The function OntoScore(i, j) returns a relevance score of node n_i with respect to k_j. The intuition behind this function is as follows. We obtain the representatives of a data source which illustrate the focus of the source, then we try to compute a kind of distance between the keyword and the representatives of the source. The shorter the distance, the

closer the keyword to the focus of the data source, and the higher the relevance.

To compute OntoScore(i, j), first, we find the representatives of data source n_i as $r_1, r_2, ..., r_m$ with weights $w_1, w_2, ..., w_m$. For each representative r_i, we compute the least common superclass of r_i with respect to the keyword k following the general definition of Learning Accuracy from Cimiano et al. [6]: $lcs(k, r_i) = \text{argmin}_{c \in Ontology}\{\delta(k, c) + \delta(r_i, c) + \delta(Root, c)\}$, where $\delta(a, b)$ is the shortest distance between node a and b in the ontology. Then, we compute the similarity score between r_i and k as follows:

$$Sim(r_i, k) = \frac{\delta(Root, f)+1}{\delta(Root, f)+1+\delta(r_i, f)+\delta(k, f)}$$

where $f = lcs(r_i, k)$. We define the ontology quality score of data source n_i with respect to keyword k_j as follows: $ontoScore(i, j) = \sum_{r_i \in n_i} w_i \times Sim(r_i, k_j)$, where m is the total number of representatives of node n_i and w_i is the weight of representative r_i. The node quality score q_{ij} = ontoScore(i, j).

Score of Matched Constraints: Some keywords represents $CountConstraints(n_i, Q) = NCM$ and $NCM < 0$. ψ represents $CountConstraints(n_i, Q) = NCM$ and $NCM >= 0$. m is the number of keywords in query Q. The above node score function considers the effect of node coverage score, node quality score, and score of matched constraints.

Node Score for Query Answering Plan: The node score of the query answering plan QAP is defined as

$$NScore(QAP) = \frac{\sum_{n_i \in QAP} NScore(n_i)}{N}$$

where N is the number of nodes in QAP. We can see that the node score of query answering plan QAP is roughly the average node score for all nodes in the plan. This function gives penalty to a plan with many nodes (longer plans with redundant nodes).

5.3 Edge Ranking Strategy

The edge score is considered as a cost one needs to pay to traverse from one node U to its descendant V . Therefore, a higher ranked edge has a lower score. An edge can be built

http://webcache.googleusercontent.com/search?q=cache:WRDbUx-VNV4J:ftp://ftp.cse.ohio-state.edu/pub/tech-report/2008/TR10.pdf+&cd=7&hl=en&ct=clnk&gl=us 10/29/2013 01:27:41 PM

sent constraints that a user sets on the query. For example, the keyword "Human" implies that the user wants to find data about humans. Each data source has a C attribute which represents the inherent constraints of the data source. We prefer to use a data source which has inherent constraints matching with the user specified constraints. The higher the number of matched constraints, the higher the node score. We use function CountConstraints(n_i, Q) to compute the score of matched constraints of a data source with respect to a query. If a query has a constraint set UC, and the node n_i has inherent constraints set NC, We compute the score of matched constraints based on the following cases: 1). UC = Φ and NC = Φ. In this case, both the query and the data source do not have any constraint, we simply simply return zero. 2). UC = Φ and NC = Φ. In this case, user does not set any constraints, but the current data source has inherent constraints. The data source inherent constraints will shrink the answers to a narrower range, as a result, the data source should receive a penalty. We return −|NC|. 3). UC = Φ and NC = Φ. In this case, user sets constraints, but data source does not have constraints. We return zero. 4). UC = Ψ and NC = Ψ and HasConfliction(UC,NC) = true. HasConfliction() is a function to detect whether there is any confliction between UC and NC. For example, user sets constraint on "Organism=Human", but the data source has constraint "Organism=Mouse". In this case, this data source definitely cannot be chosen for this query. So we will return a special value null. 5). UC = Φ and NC = Φ and HasConfliction(UC,NC) = false. In this case, we return |UC ∩ NC|−|NC − UC|. |UC ∩ NC| is the number of user specified constraints which are also matched in data source constraint set. |NC − UC| is the number of data source constraints which are not requested by the user.

Node Score for A Single Node: Now we define the node score for a node n_i as

$$\text{NScore}(n_i) = \begin{cases} \text{null} & \text{if } \phi, \\ \sum_{k=2}^{m} \text{NScore}(n_i, k) & \text{if } \chi, \\ \sum_{k=2}^{m=(|NC|+1)} \text{NScore}(n_i, k)\times(|NC|+1) & \text{if } \psi. \end{cases}$$

ϕ represents CountConstraints(n_i, Q) = null. γ represents CountConstraints(n_i, Q) = k and $k \le |UC|$. ψ represents CountConstraints(n_i, Q) = k and $k > |UC|$. m is the number of matched constraints between UC and NC. NScore(n_i, k) is the score when n_i matches k constraints. We can see that the higher the number of matched constraints, the higher the node score. Also, if a node matches more constraints than the number of constraints specified by the user, it will receive a penalty.

Edge Score: Edge score is used to measure the importance of an edge. Intuitively, if two nodes are connected by a first type dependence relation, the edge should be ranked higher than the edge connecting two nodes with second or third type dependence relation. Therefore, the edge connecting two nodes with first type dependence relation should have a lower score. An edge can be built between U and V and a constant score is assigned to it only when there is a first type dependence relation between them. According to the desired properties in Section 5.1, if there exists second and/or third type dependence relation, we reduce the edge score, essentially giving it a bonus. Along this line, the edge score of e connecting U and V can be defined as

$$\text{EScore}(U, V) = \begin{cases} \infty & \text{if no first type dependence relation between U and V,} \\ 4 - T1 - T2 - T3 & \text{if has first type dependence relation between U and V.} \end{cases}$$

$$T1 = \begin{cases} 0 & \text{if no second type dependence relation between U and V,} \\ 1 & \text{if has second type dependence relation between U and V.} \end{cases}$$

$$T2 = \begin{cases} 0 & \text{if no third type dependence relation between U and V,} \\ 1 & \text{if has third type dependence relation between U and V} \\ & \text{and it is pointing from optional input to must-fill input.} \end{cases}$$

$$T3 = \begin{cases} 0 & \text{if no third type dependence relation between U and V,} \\ 1 & \text{if has third type dependence relation between U and V} \\ & \text{and it is pointing from optional input to optional input.} \end{cases}$$

The total edge score for query answering plan QAP is defined as:

$$\text{EScore}(QAP) = \frac{1}{\sum_{e \in QAP} \text{EScore}(U, V)}$$

5.4 Query Answering Plan Ranking Strategy

Combining the node and edge ranking functions above, the query answering plan ranking function is a linear combination of its node score and edge score, which is defined as :

$$\text{Score}(QAP) = \lambda \times \text{NScore}(QAP) + (1-\lambda) \times \text{EScore}(QAP)$$

In our current system, we set the parameter λ to be 0.5. We prefer query answering plans that have a higher score.

http://webcache.googleusercontent.com/search?q=cache:WRDbUx-VNV4J:ftp://ftp.cse.ohio-state.edu/pub/tech-report/2008/TR10.pdf+&cd=7&hl=en&ct=clnk&gl=us 10/29/2013 01:27:41 PM

(a) (b)

Figure 3: Comparison among BIA, NA and EXA: (a) Comparison between BIA and NA;(b) Comparison between BIA and EXA.

Table 2: Query Statistics

Query ID	Number of Terms
1-10	2-5
11-18	8-12
19-24	17-23
25-28	27-33
29,30	37-43

6. EVALUATION

This section describes the experiments we conducted to evaluate our algorithm.

6.1 Experiment Setup

Our evaluation was done using 14 different biological deep web databases we have integrated. We created 30 queries for our evaluation. Among these 30 queries, 10 are real queries specified by a domain expert we have collaborated with. The remaining 20 queries were generated by randomly selecting query keywords from the ontology to form queries. We create two types of queries, keyword-attributes queries and keyword-keyword relation queries. Among the 30 queries, 22 queries are of the first type, and 8 queries are of the second type. We also vary the number of terms in each query in order to evaluate the scalability of our algorithm. Table 2 summarizes the statistics for the 30 queries.

Among our queries, we have several queries with a large number of keywords. We choose these queries for the following reasons. Unlike the traditional relational database queries in commercial domains, a biological domain query is very likely to have a large number, i.e., 20 or more, keywords. This is because users tend to use high-level abstract terms in their keyword search. For example, SNP Frequency, a very common query keyword, is a high-level abstract term that corresponds to four low-level keywords. Another reason for creating long queries was that we wanted to test our algorithm in extreme cases.

In the evaluation, we compare our BIdirectional Algorithm (BIA) with three other algorithms, which are the Naive Algorithm (NA), the EXhaustive Algorithm (EXA) and the Backward Algorithm (BA).

Naive Algorithm: As the name suggests, this algorithm [illegible] all data sources which can be queried at each round, until all keywords are covered. This algorithm can quickly find a query answering plan, but it is likely to have a very low score and a long execution time.

Exhaustive Algorithm: This algorithm searches the entire space in a recursive manner, and compares every possible query answering plan. Then, it selects the plan with the highest score. This algorithm always finds the optimal answering plan, but has high time and space requirements.

Backward Algorithm: This algorithm uses exactly the same data structures as the bidirectional planning algorithm. The only difference is that backward algorithm can only search from the backward direction.

6.2 Evaluation Metrics

Query Answering Plan Score: We use the ranking function introduced in Section 5 to compute a score for each query answering plan. We prefer the plan with a higher score, because higher score implies that a smaller number of data sources are involved, and they have higher relevance.

Query Answering Plan Estimated Execution Time: The query execution time of a deep web data source is estimated by issuing multiple randomly selected sample queries. Since the query answering plan has a disconnected component and some sources can be executed in parallel, the estimated execution time of the entire plan is computed based on the parallel execution model. The longest path (in terms of time) in the plan determines the execution time.

Query Planning Time: Efficiency of query planning algorithm is an important consideration. We record the query planning algorithm's running time.

Planning Time for Generating the First Query Plan: BIA and BA algorithms can generate multiple query answering plans. We record the time used to generate the first (possibly the best) plan. We prefer the algorithm which can generate the first plan quickly.

Average Query Planning Time: For BIA and BA, we compute the average query planning time by dividing the total query planning time by the number of query plans generated.

In addition to comparing different algorithms using the above metrics, we have also evaluated the scalability of the bidirectional planning algorithm with respect to the number of data sources involved in the query.

http://webcache.googleusercontent.com/search?q=cache:WRDbUx-VNV4J:ftp://ftp.cse.ohio-state.edu/pub/tech-report/2008/TR10.pdf+&cd=7&hl=en&ct=clnk&gl=us 10/29/2013 01:27:41 PM

does query planning in a naive way. The algorithm selects

Table 3: Query Planning Time Comparison Between BIA, NA and EXA

Selected Query	NA Planning Time (ms)	EXA Planning Time (ms)	BIA Planning Time (ms)
1	6	1385	30
2	3	774	29
3	5	1020	131
4	15	3090	62
5	4	615	48

high score does not necessarily have a low estimated execution time, and vice versa. The reason is that a very highly ranked data source may have longer execution time. As a result, it is possible to see a query answering plan with high score but a large execution time.

6.3 Experiment Results

6.3.1 Comparing BIA against NA and EXA

In Figure 3, sub-figure (a) shows the comparison between BIA and NA. It is plotted in logarithmic scale. The SRatio (Diamond) is the ratio between BIA's generated query answering plan's score and NA's generated query answering plan's score. The ETRatio (Rectangle) is the ratio between NA's generated plan's estimated execution time and BIA's generated plan's estimated execution time. Figure (b) shows the same comparison between BIA and EXA. In both the sub-figures (a) and (b), the x-axis is the query ID, and the y-axis is the ratio value.

From sub-figure (a), we can see that except for queries 1 and 4, the plans generated by BIA always have much higher (more than 5 times) score than the plans generated by NA. Similarly, the execution time of the plans generated by BIA are always lower than the execution time of plans generated by NA. For queries 1 and 4, NA can also obtain very good results. This is because these two queries are very simple, and only need a single data source. For the query 4, the execution time of NA's query plan is even shorter than that of BIA's query plan. This is reasonable, because a data source with higher relevance may have long execution time.

From sub-figure (b), we can observe that the score of plans generated by BIA are almost the same as the score of the plans generated by EXA. This shows that the quality of the plans generated by BIA is very close to the quality of the

One important observation is that a query plan with a

Figure 5: The Scalability of BIA according to Number of Data Sources Involved in Queries.

6.3.2 Comparing BIA against BA

In Figure 4, sub-figure (a) shows the SRatio and ETRatio between BIA and BA. Because BIA and BA use the same data structure, we expect both the ratios should be near 1. We can observe that this is actually the case. We can also note that for some queries (10 out of 30), BIA obtains answers with higher scores than BA. This shows that in terms of the quality of query answering plan, BIA and BA have nearly the same performance, with BIA outperforming BA in some cases.

Because BA searches only from one direction, we expect that BIA will beat BA in terms of query planning time. Sub-figure (b) and (c) are plotted in logarithmic scale. In Figure 4, sub-figure (b) shows the ratio between BA's average query planning time and BIA's average query planning time. We can observe that for most queries, BIA takes smaller amount of time to generate a query plan. The sub-figure (c) shows the time ratio for the first generated query plan between BA and BIA. We have the same observation, i.e. BIA can generate the first query plan much faster than BA.

6.3.3 The Scalability of BIA:

From Figure 5, we can observe that in terms of the average planning time, there is a sharp increase when the number of data sources increases from 2 to 4. Then, the planning time increases moderately with respect to the number of data sources. In terms of the first plan generation time, there is a sharp increase when the number of data sources increases from 7 to 10, otherwise, the increase is moderate. This shows that our system has good scalability.

http://webcache.googleusercontent.com/search?q=cache:WRDbUx-VNV4J:ftp://ftp.cse.ohio-state.edu/pub/tech-report/2008/TR10.pdf+&cd=7&hl=en&ct=clnk&gl=us 10/29/2013 01:27:41 PM

plans generated by BIA is very close to the quality of the optimal plans generated by EXA. The execution time has the same pattern as the score, except for queries 5 and 7. For these two queries, we can see that the estimated execution time for query plans of BIA are even shorter than the optimal query plan's execution time. This is reasonable because our optimal query plan is generated based on query answering plan score and BIA algorithm, for this two queries, selects the data source with a lower execution time.

In Table 3 we show the comparison of query planning time between BIA, NA, and EXA. We can clearly see that BIA takes much less time than EXA. Compared to the time of NA, BIA's running time is still modest.

In summary, BIA outperforms NA in terms of generated plan's quality. BIA can generate nearly optimal query answering plans while take much less time than EXA.

This shows that our system has good scalability.

6.3.4 Actual Query Result Evaluation of BIA

The answers retrieved by our system were checked by a biologist collaborating with us. Currently, we have wrappers for 8 deep web data sources, out of the 14 we used for query planning. For the plans that only extract data from these 8 sources, the plans are automatically executed and answers are retrieved and tabulated. For all other plans, the answers were extracted manually. Both automatically and manually retrieved answers were presented to the biologist.

From the feedback from the domain scientist, all answers to the 30 experimental queries are correct and sufficient, with the exception for one query. The query is "rs7412, rs12982192", which is intended to find the relationship between the two SNPs. The expected answer was that the two SNPs are located in the gene APOE and the chromosome 19. Our system can correctly find the first relation-

(a) (b) (c)

Figure 4: Comparison between BIA and BA: (a) Query Plan Comparison between BIA and BA;(b) Average Planning Time Ratio between BA and BIA;(c) First Plan Generating Time Ratio between BA and BIA.

ship. For the second relationship, our system indeed finds a correct query answering plan, but because the data source our system chose has incomplete data (rs12982192 are not in its database), the relationship is not discovered. This limitation can be addressed in the future by having additional knowledge about the data sources. Our system can also enable execution of another query plan on request from the domain scientist, which may retrieve data from different sources.

7. RELATED WORK

We now compare our work with existing work on a number of related topics, including query planning, keyword search do query planning and for answering the query.

Keyword Search on Relational Databases: Recently, performing keyword based search over relational databases has attracted a lot of attention [14, 13, 17, 22, 1, 2, 23]. In relational database keyword search, databases are represented as graphs. Each row in relation table is represented as a node, and foreign keys are represented as edges. The graph stores the actual data in the databases. The critical difference in our algorithm are as follows. We also represent the entire deep web as a graph. The nodes in our graph are deep web data source query schemas, and as a result, our graph model only contains the high level abstract information for each data source. We do not know the actual