A Journal of Software Engineering and Applications, **2012, 5, 88-93**
doi:10.4236/jsea.2012.512b018 Published Online December 2012 (http://www.scirp.org/journal/jsea)

Evaluating Relational Ranking Queries Involving Both Text Attributes and Numeric Attributes

Liang Zhu[1], Zhaoliang Xie[1], Qin Ma[2]

[1]Key Laboratory of Machine Learning and Computational Intelligence, School of Mathematics and Computer Science, Hebei University, Baoding, Hebei 071002, China; [2]Department of Foreign Language Teaching and Research, Hebei University, Baoding, Hebei 071002, China
Email: zhu@hbu.edu.cn, xiezhaoliang533@163.com, maqin@hbu.edu.cn

Received October 12, 2012

ABSTRACT

In many database applications, ranking queries may reference both text and numeric attributes, where the ranking functions are based on both semantic distances/similarities for text attributes and numeric distances for numeric attributes. In this paper, we propose a new method for evaluating such type of ranking queries over a relational database. By statistics and training, this method builds a mechanism that combines the semantic and numeric distances, and the mechanism can be used to balance the effects of text attributes and numeric attributes on matching a given query and tuples in database search. The basic idea of the method is to create an index based on WordNet to expand the tuple words semantically for text attributes and on the information of numeric attributes. The candidate results for a query are retrieved by the index and a simple SQL selection statement, and then top-*N* answers are obtained. The results of extensive experiments indicate that the performance of this new strategy is efficient and effective.

Keywords: Relational Database; Ranking Query; Semantic Distance; Numeric Distance; WordNet

1. Introduction

A relational ranking query (or top-*N* query) is to find the *N* tuples that satisfy the query condition the best but not necessarily completely, and the results are sorted according to a given ranking function. Researches on top-*N* queries have intensified since late 1990s [1-3], and most of the researches involve numeric attributes and use a numeric distance function (say, L_p-*norm* distances, p =1, 2, and ∞) to reduce a massive result set of a conventional query to a few of the most relevant answers. However, there are many applications where ranking queries involving both text attributes and numeric attributes are available for processing.

Example 1. Consider a database of used books with schema: ***Books***(*id#*, *title*, *price*, *year*,…). Suppose a customer wants to buy a used book with *title* on "*film*", *price* around "$50" and *year* about "2000", where *title* is a text attribute with semantics, and *price* and *year* are two numeric attributes. Obviously, a book on "*algebra*" with *price* = "$50" and *year* = "2000" is not the desired result for the customer though the values for *price* and *year* exactly match that of the query respectively. Another book on "*movie*", however, with *price* = "$53" and *year* = "2001" maybe satisfy the need of the customer.

Ipum99

idx	*A29*	*A40*	*A50*
87197	42	2	20000
6505	28	3	25000
80789	**51**	**930**	**33000**
10860	9	999	999999

Occ50

num	*occ50*
2	Airplane pilots and navigators
3	Architects
930	**Gardeners, except farm, and groundskeepers**
999	N/A (blank)

Figure 1. Parts of IPUMS Census Database.

Example 2. As shown in **Figure 1**, database **IPUMS** has two relations ***Ipum99*** and ***Occ50***, which come from [4]. In relation ***Occ50***(*num*, *occ50*), its primary key *num* is the value label of occupation1950. Relation ***Ipum99*** has 61 numeric attributes where *A29* is age, *A50* means income, and *A40* is the foreign key referencing ***Occ50***.*num*. Furthermore, ***Ipum99*** is added an attribute *idx* by us as identifier. A user is looking for the information of "*a horticulturist with age about 50*" from **IPUMS**. Since there is no such word "*horticulturist*" in ***Occ50***.*occ50*, the answer will be nothing by using the

traditional SQL selection statement. In fact, there is a tuple ("*Gardeners, except farm, and groundskeepers*", *age* = 51, …) (with gray color) in **IPUMS** may be an answer for the user.

WordNet::Similarity [5] is an open source Perl module for measuring the semantic distance/similarity between two *words*; however, it is not easy for us to use the source directly to evaluate ranking queries [6].

Researches on semantic search in IR and SW (semantic web) have gained attention since 1990s. Taking thesaurus ontology navigation as a step in *query expansion*, many query expansion techniques are employed in keyword search, say, query terms are expanded to WordNet synonyms and meronyms, using the Boolean OR operations available in most web search engines [7]. Different from the above query expansion techniques in IR and SW, we discuss semantic match between query words and tuple words *in relational databases* via *tuple expansion*.

In recent years, ontologies have been used to build applications in database community [8, 9]; however, it is a challenging job to construct efficient ontologies [9, 6]. Our method is different from the ontology-based techniques. Firstly, instead of dealing with the challenge of constructing ontology, we create a simple table maintained by RDBMS, and store information of index and ranking function into the table. Secondly, our strategy is more general since it is based on WordNet. Finally, our techniques are more efficient than the above ontology-based methods due to the efficiency of RDBMS.

Another different yet related research topic is keyword search in relational databases [10-12], which supports free-form keyword search in relational databases without necessarily requiring the users to know the database schema. Keyword search may be suitable for any text attribute, but it is exact search without dealing with semantic match. For instance, it cannot return the books on "*movie*" for the query keyword "*film*". If query words match exactly tuple words, the results of semantic search will contain those of keyword search, and improve the effectiveness of keyword search by using our method.

There are two challenging problems for evaluating the type of relational ranking queries in this paper. The first is how to design a good semantic distance function that measure the semantic similarity between the query words and the tuple words, [6] presented a solution for this problem. The second is how to combine the semantic and numeric distances. We employ statistics and training to solve the problem, and create an index to process ranking queries in terms of semantic and numeric matching in database search. Moreover, this work is a continuation of the work in [6], which studied the processing of relational ranking queries only with text attributes, without numeric attributes.

2. Problem Definition and Ranking Function

Assume that $\boldsymbol{R_0}(idx, A, B_1, B_2, \ldots, B_m, \ldots)$ is a relation with identifier *idx*, where A is a text attribute with natural language semantics and $B_1, B_2, \ldots, B_m$ are m numeric attributes. Using the project operation $\pi_A(\boldsymbol{R_0})$ (duplicate tuples are eliminated), we get a new relation $\boldsymbol{R}(tid, A)$ with identifier *tid*. Furthermore, $\boldsymbol{R_0}$ is added by an attribute *FKid* that is the foreign key referencing $\boldsymbol{R}.tid$, and then the schema of $\boldsymbol{R_0}$ becomes $\boldsymbol{R_0}(idx, A, B_1, B_2, \ldots, B_m, \ldots, FKid)$. Or, let $\boldsymbol{R}(tid, A)$ and $\boldsymbol{S}(idx, B_1, B_2, \ldots, B_m, \ldots, FKid)$ be two relations and $\boldsymbol{S}.FKid$ be the foreign key referencing $\boldsymbol{R}.tid$, then $\boldsymbol{R_0} = \boldsymbol{R} \bowtie \boldsymbol{S}$ with $\boldsymbol{S}.FKid = \boldsymbol{R}.tid$.

Let $\boldsymbol{t}$ be a tuple in $\boldsymbol{R_0}$, then $\boldsymbol{t}[A] = (tw_1, tw_2, \ldots, tw_n)$ is the word-string with n words on the text attribute A, and $\boldsymbol{t}[B_j] = b_j$ is the numeric value on the attribute B_j ($1 \le j \le m$). For simplicity, we denote $\boldsymbol{t} = (\boldsymbol{t}_A, b_1, b_2, \ldots, b_m)$ where $\boldsymbol{t}_A = (tw_1, tw_2, \ldots, tw_n)$, and call tw_i a tuple word and b_j a tuple value ($1 \le i \le n$, $1 \le j \le m$).

As defined in [6], given a tuple word w, its ***kinship words*** include the five kinds of words in WordNet: (1) word w itself, (2) morph, (3) synonyms, (4) the immediate hyponyms (subordinates), and (5) the immediate hypernyms (superordinates). The set of all kinship words of w is denoted by $\mathcal{K}(w)$. For instance, the kinship word set of "*computers*" is $\mathcal{K}$(*computers*) ={*computers*, *computer*, *data processor*, *machine*, *internet site*, *calculator*, …}.

Consider a ranking query $\boldsymbol{q} = (\boldsymbol{q}_A, q_1, q_2, \ldots, q_m)$, where $\boldsymbol{q}_A = (qw_1, qw_2, \ldots, qw_k)$ is a word-string with k query words, and q_j is a numeric value ($1 \le j \le m$). For tuple $\boldsymbol{t} = (\boldsymbol{t}_A, b_1, b_2, \ldots, b_m)$ in $\boldsymbol{R_0}$, denote $p_j = |q_j - b_j|$ ($1 \le j \le m$), and $d_A = d(\boldsymbol{q}_A, \boldsymbol{t}_A)$ that is the semantic distance between $\boldsymbol{q}_A$ and $\boldsymbol{t}_A$ defined by Definition 1 to 3 in [6]. Moreover, d_A belongs to the interval (0, 1].

We need to find a mechanism combining the semantic and numeric distances d_A and p_js, which can be used to evaluate the query $\boldsymbol{q}$ over the relation $\boldsymbol{R_0}$, i.e., to define a ranking function $d(\boldsymbol{q}, \boldsymbol{t}) = \psi(d_A, p_1, p_2, \ldots, p_m)$.

Intuitively, the ranking function $d(\boldsymbol{q}, \boldsymbol{t})$ should satisfy the following: First, a smaller $d(\boldsymbol{q}, \boldsymbol{t})$ indicates closer the pair $(\boldsymbol{q}, \boldsymbol{t})$. Second, $d(\boldsymbol{q}, \boldsymbol{t})$ needs to balance the effects of d_A and p_js in matching $\boldsymbol{q}$ with $\boldsymbol{t}$. Finally, it should be easy to implement.

To obtain $d(\boldsymbol{q}, \boldsymbol{t})$, we use the statistics of the domains of $B_1, B_2, \ldots, B_m$ and training.

Since the semantic distance $d_A \in (0, 1]$, we normalize it by scientific notation, $d_A = \alpha \times 10^{-h}$, where $1.0 \le \alpha < 10.0$, and h is a nonnegative number (i.e., $h \ge 0$, if $h = 0$,

we define $-0 = 0$), say, $d_A = 0.001041 = 1.041\times10^{-3}$. We will see that h (the absolute value of the exponent) plays an important role in the ranking function $d(\boldsymbol{q}, \boldsymbol{t})$.

In collecting statistics, there is a step of cleaning data and removing outlier, and then we get $Min(B_j)$ and $Max(B_j)$ of numeric attribute B_j ($1 \le j \le m$). Based on the semantics of attribute B_j, we obtain its reasonable unit $Z_j > 0$, say, $Z_{year} = 1$ for attribute *year* and $Z_{price} = 0.01$ for *price* respectively in database **BOOK**.

Let $M_j = Max(B_j) - Min(B_j) = \beta_j \times 10^{c_j}$, where $1.0 \le \beta_j < 10.0$ ($M_j \ge 0$, obviously. If $M_j = 0$, let $M_j := Z_j$ then $M_j > 0$). If $M_j \ge 1$, we have $c_j \ge 0$, else if $0 < M_j < 1$, let $M_j := M_j / Z_j$, and $p_j := p_j / Z_j$, then $M_j \ge 1$ and $c_j \ge 0$.

Let $e_j := c_j + 1 \ge 1$, for $1 \le j \le m$. By using the statistics and training in our experiments, we obtain the ranking function $d(\boldsymbol{q}, \boldsymbol{t}) = \psi(d_A, p_1, p_2, \ldots, p_m)$ below.

$$d(\boldsymbol{q}, \boldsymbol{t}) = \quad d_A + \Sigma_{j=1}^{m} (p_j/M_j)^{h/e_j}$$

For example, in our experiments for relation ***Books*** (*title*, *price*, *year*,…) with 56180 tuples. *price* is in (0, 1000] (except 633 tuples with *price* > 1000), and *year* belongs to [1958, 2008] (except 651 tuples with *year* < 1958). Let attribute A, B_1 and B_2 be *title*, *price* and *year* respectively, then $M_1 = 1000 - 0 = 1000 = 1.0\times10^3$, $M_2 = 2008 - 1958 = 50 = 5.0\times10^1$, and then $c_1 = 3$, $c_2 = 1$, $e_1 = c_1+1 = 4$ and $e_2 = c_2+1=2$. Thus, $d(\boldsymbol{q}, \boldsymbol{t}) = d_A + (p_1/1000)^{h/4} + (p_2/50)^{h/2}$.

For query $\boldsymbol{q}$, we will return *dis-N* tuples defined in [6] to replace *top-N* ones. Let $\boldsymbol{T}$ be a set of tuples, a tuple $\boldsymbol{t} \in \boldsymbol{T}$ is called a *dis-N* tuple of $\boldsymbol{q}$, if $d(\boldsymbol{q}, \boldsymbol{t}) \le \min_N \{d(\boldsymbol{q}, \boldsymbol{t}_i) \mid \boldsymbol{t}_i \in \boldsymbol{T}\}$, which means the Nth minimum value in the set $\{d(\boldsymbol{q}, \boldsymbol{t}_i) \mid \boldsymbol{t}_i \in \boldsymbol{T}\}$.

3. Creation of sn-Index

We extend w-index in [6] to sn-index (stands for *s*emantic and *n*umeric index) in this section.

3.1. Information Stored in sn-Index

A relation is employed to store the information in sn-index, which is called ***IndexTable*** with schema ***IndexTable***(*id#*, *Word*, *Size*, *dbNSize*, *DBValue*, *BValue*, *bSize*), say, (2490, *nights*, 2, 2, '47898,0,3,1;2413,0,…', '47898,8185,10.80,1997;…', 3) is a tuple in ***IndexTable*** for database **BOOK**.

The attribute *Word* indicates the kinship word of tuple words, and the relevant information will be stored in the attribute *DBValue*. The value of *DBValue* is a string with form "*tid*,*d*,*n*,*f* ;*tid*,*d*,*n*,*f*;…;" where "*tid*,*d*,*n*, *f* ;" is a node, $d = 0, 1, \ldots, 5$, "$d = i$" is the subscript i of semantic distance d_i defined by Definition 1 in [6] , n is the number of tuple words in the tuple with *tid*, and f is the frequency in the set of kinship words of the tuple. The attribute *Size* is the number of *tid*s associated with *kw*, and the duplicate *tid*s are counted repeatedly. The attribute *dbNSize* means the number of nodes in *DBValue*.

Attribute *BValue* is a string as "*tid*,*idx*,b_1,b_2,…,b_m; *tid*,*idx*,b_1,b_2,…,b_m;…;", where " *tid*,*idx*,b_1,b_2,…,b_m;" is a node, *tid* is the identifier of a tuple in the relation $\boldsymbol{R}$(*tid*, *A*), and *idx* is that in $\boldsymbol{R_0}$(*idx*, …). The attribute *bSize* means the number of nodes in *BValue*.

3.2. Procedure of Creating sn-Index

Reconstructing w-index in [6], we obtain sn-index as shown in Figure 2. Sn-index consists of four parts: (1) a hash table with a hash function h(), its each bucket contains a pointer *pWnPointer* which points to a node in wn-list; (2) one wn-list, its node has structure {*iRow*, *kw*, *size*, *dbNSize*, *bSize*, *pDBList*, *pBList*}; (3) s db-lists, the structure of the node is {*tid*, *d*, *n*, *f*}; (4) s b-lists, node structure {*tid*, *idx*, b_1, b_2, …, b_m}. The main difference between sn-index and w-index in [6] is that b-lists (i.e., B_{ij}'s in Figure 2) are added. The algorithm of creating sn-index is shown as follows:

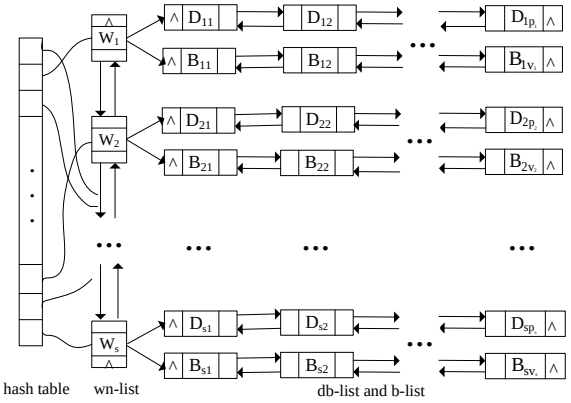

Figure 2. Structure of sn-index.

Step 1. For the special character string $\boldsymbol{t}_A = (tw_1, \ldots, tw_n)$, calculate its hash value $h = h(\boldsymbol{t}_A)$; check the pointer *pWnPointer* in the bucket of the hash table, and wn-list,

(1) If *pWnPointer* is NULL, or (tw_1, …, tw_n) is not in wn-list,

(a) Create a new db-list, and a dbNode as $tid = \boldsymbol{t}_A.tid$, $d := -1$ (means the special case), $n := 1$ and $f := 1$. Insert the dbNode into the db-list.

(b) Using *tid* above, search (idx, b_1, b_2, …, b_m) from $\boldsymbol{R_0}$ (or from $\boldsymbol{S}$) by the following the SQL selection statement

Select idx, B_1, B_2, …, B_m *from* $\mathbf{R_0}$ *where* FKid = tid

Let z be the size of the result set {(idx, b_1, b_2, …, b_m)}, Create a new b-list, and z bNode(s), insert the bNode(s) into the b-list.

(c) Create a wnNode for wn-list, kw:= (tw_1, …, tw_n), and $dbNSize$:=1 and $bSize$:= z. Insert the wnNode into wn-list according to the alphabet order of kw.

(d) Get the pointers $pDBList$, $pBList$ and $pWnPointer$.

(2) Else if the string $\boldsymbol{t}$ is found out in wn-list, then increase $dbNSize$ by 1 in the wnNode with $\boldsymbol{t}$, create a dbNode as above, and let it be the first node of the db-list. Additionally, by the SQL selection statement as (b) above, get a result set {(idx, b_1, b_2, …, b_m)} with size $z \geq 1$. Create z bNode(s), insert the node(s) into the b-list, and increase $bSize$ by z.

Step 2. For each tuple word $tw_i \in \boldsymbol{t}$, and each kinship word kw of tw_i, get h=h(kw), check $pWnPointer$ in the bucket of the hash table, and wn-list,

(1) If $pWnPointer$ is NULL, or kw is not in wn-list, do the same jobs as the above (1) in Step1 except for replacing $\boldsymbol{t}_A$ by kw and defining $d = k$, $k \in \{0, 1, \ldots, 5\}$ such that d_k is d(kw, tw_i) in Definition 1 in [6].

(2) Else if kw has been in wn-list, then get its db-list and b-list. There are two cases.

Case 1, in the db-list, there is a dbNode with dbNode.tid = $\boldsymbol{t}$.tid, replacing d by the smaller if the two distances are different, and increasing f by 1, that is OK.

Case 2, if no dbNode in the db-list satisfies dbNode.tid = $\boldsymbol{t}$.tid, increase $dbNSize$ by 1 in the wnNode with kw, create and insert a new dbNode into the db-list according to the increasing order of distance d. In addition, by the SQL selection statement as (b) above, get a result set {(idx, b_1, b_2, …, b_m)} with size $z \geq 1$. Create z bNode(s), insert the node(s) into the b-list, and $bSize$:= z.

Step 3. Storing sn-index.

To evaluate query $\boldsymbol{q}$, we use two storing strategies. Strategy-1, the entire sn-index is in main memory. Strategy-2 will store db-lists and b-lists in fixed disk and only load the hash table and wn-list into main memory.

4. Evaluation of Ranking Query

For query $\boldsymbol{q}$ = ($\boldsymbol{q}_A$, q_1, q_2, …, q_m), firstly, matching the query words with kinship words of $\boldsymbol{R}$(tid, A) via sn-index, we obtain the set $\boldsymbol{T}$ (= {tid}) of identifiers of candidate tuples, and compute the semantic distances between $\boldsymbol{q}$ and its candidate tuples, and then get the set $\boldsymbol{L}$ of information of numeric attributes by each identifier tid in $\boldsymbol{T}$; secondly, compute $d(\boldsymbol{q}, \boldsymbol{t}) = \psi(d_A, p_1, p_2, \ldots, p_m)$ between $\boldsymbol{q}$ and each of its candidate tuples, and then obtain {idx}, which is the sorted set of identifiers of dis-N results according to $d(\boldsymbol{q}, \boldsymbol{t})$; finally, we retrieve the dis-N tupels from underlying database and display the ranked answers.

To get the set $\boldsymbol{T}$ is an important step for query processing. The intermediate results are stored in a temporary list, denoted by T-List. Its node has the structure {tid, d[K], d_A} where K is the maximum number of query words for $\boldsymbol{q}_A$ = (qw_1,…, qw_k). If $k > K$, let $\boldsymbol{q}_A$ = (qw_1, …, qw_K) (K = 30 in our implementations). The intermediate results for obtaining set $\boldsymbol{L}$ are saved in a temporary list L-List, and the structure of its node is {d_A, p_1, p_2, …, p_m, d, tid, idx}

We discuss Strategy-2 of storage first (i.e., we only load the hash table and wn-list in main memory).

(1) Normalization of $\boldsymbol{q}_A$. We remove some symbols, character strings or stop words, and replace some of them by normal strings for $\boldsymbol{q}_A$. Denoted again by $\boldsymbol{q}_A$ = (qw_1, …, qw_k).

(2) For each $qw_i \in \boldsymbol{q}_A$ (i= 1, …, k), calculate hash value h=h(qw_i), and check qw_i in wn-list. If qw_i is found out, we get the wnNode in wn-list, and then use the SQL selection statement

select ∗ *from* IndexTable *where* id# = wnNode.iRow

to obtain dbNode(s). For each dbNode, we save the values into T-List, and compute d_A= $d(\boldsymbol{q}_A, \boldsymbol{t}_A)$, then obtain $\boldsymbol{T}$.

(3) If above $\boldsymbol{T} \neq \varnothing$, for each $tid \in \boldsymbol{T}$, we get its corresponding bNode(s) by using select statements from ***Indextable***, store them into L-List, compute $d := \psi(d_A, p_1, \ldots, p_m)$, and then obtain the set $\boldsymbol{L}$ (={idx}) of candidate dis-N tuple identifiers.

(4) Given a positive integer N, we get the set of dis-N tuple identifiers $\boldsymbol{L}_N \subseteq \boldsymbol{L}$; then, we obtain the dis-N tuples of the query $\boldsymbol{q}$ by using the SQL selection statement as the following format:

Select R0.*, R.A *from* R0, R *where* (R0.FKid = R.tid)
and (R.idx in $\boldsymbol{L}_N$) *and* …

(5) Display the dis-N tuples sorted by $d(\boldsymbol{q}, \boldsymbol{t})$.

Next, we discuss Strategy-1 of storage, i.e., the entire w-index is saved into main memory. The query processing is similar to the above situations, except for steps (2) and (3), we can get dbNode(s) and bNode(s) from sn-index directly, without using select statements. Thus, the response time of Strategy-1 will be smaller than that of Strategy-2.

Example 2 (cont.). For the dis-5 query "*a horticulturist with age=50 and income=30000*" against **IPUMS**, the answers with form (idx, $\boldsymbol{t}_A$, age, $income$, $d(\boldsymbol{q}, \boldsymbol{t})$) are

$\boldsymbol{t}_1$: (80789, $\boldsymbol{t}_A$, 51, 33000, 0.609496),
$\boldsymbol{t}_2$: (07296, $\boldsymbol{t}_A$, 48, 25000, 0.729997),
$\boldsymbol{t}_3$: (77380, $\boldsymbol{t}_A$, 37, 30600, 0.747815),

t_4: (43851, t_A, 51, 21894, 0.763280), and
t_5: (73792, t_A, 47, 25000, 0.766698),
where t_A = t[*occ50*] = "*Gardeners，except farm, and groundskeepers*".

5. Experimental results

Our experiments are carried out using Microsoft's SQL Server 2000 and VC++6.0 on a PC with Windows XP, Intel(R) Core(TM) i5-2400/3.10GHz 3.09GHz CPU, and 2.98GB memory. In addition, WordNet 2.1 and its API functions, ODBC, and ODBC API functions are used.

Datasets: We use two real datasets. The first one is **IPUMS** with two relations that is a fragment of US Census Bureau data [4]. The relation ***Occ50***(*num*, *occ50*) contains 286 tuples with 2 attributes. The relation ***Ipum99***(*idx*, *FKnum*, *age*, *income*, ...) has 61 attributes and 88443 tuples, where *age* is *A29*, *income* is *A50*, and *FKnum* is *A40* that is the foreign key referencing ***Occ50***.*num*. $\boldsymbol{R_0}$ = ***Ipum99*** ⋈ ***Occ50*** with *FKnum*= *num*. Part of **IPUMS** is shown in Example 2 in Section 1.

The second dataset **BOOK** comes from the Library at Beijing University of Technology, which is the record set of English books in the Library, and produces two relations. One is ***Books***(*id#*, *isbn*, *title*, *author*, *publisher*, *price*, *year*, *FKid*) having 56180 tuples. The other relation ***Titles***(*tid*, *title*) has 48107 tuples (duplicate titles are removed). In addition, ***Books*** and ***Titles*** act as $\boldsymbol{R_0}$ and ***R*** respectively. ***Books***.*id#* corresponds to $\boldsymbol{R_0}$.*idx*, and ***Books***.*FKid* is the foreign key referencing ***Titles***.*tid*.

Attributes *tid*, *A*, B_1, and B_2 described in Section 2 correspond *num*, *occ50*, *age* and *income* for **IPUMS**, and *tid*, *title*, *price* and *year* for **BOOK**, respectively.

Space cost of sn-index: Strategy-1 and Strategy-2 are used for **IPUMS** and **BOOK** respectively. The main memory space costs are: *index-space*-IPUMS is about 3.6MB, and *index-space*-BOOK about 3MB.

Workloads: We build a program to create a workload that is a set of 100 queries for each database. First, we choose 100 tuples from ***R*** randomly, and then for each tuple ***t***, select 1~10 kinship word(s) from $\mathcal{K}(\boldsymbol{t})$ randomly, where the numbers of simple queries with 1~3 words and complex queries with 4~10 words are both 50. We classify them into 10 groups G_i (i =1, 2, …, 10), and the query in G_i has i search word(s). The size of G_i is random.

For the legends in the following figures, the suffixes "1", "2", …, and "100" indicate the dis-1, dis-2, …, and dis-100 queries, respectively.

5.1. Elapsed Time

Figure 3 illustrates the average elapsed time for executing all queries in each G_i for **IPUMS** by fourteen curves *it*N's and *dt*N's, which stand for *index-time*'s and *DB-time*'s respectively. The seven curves *it*1 to *it*100 are (almost) the same, and are related to queries, but not to N (N = 1, 2, …, 100), which are from 11 to 230 milliseconds. The other seven curves *dt*1 to *dt*100 are the costs of retrieving tuples from DB by using SQL selection statements for the natural join of ***Ipum99*** and ***Occ50***, which are smaller than 600 milliseconds.

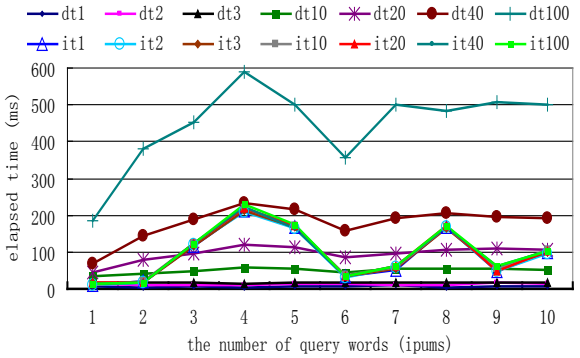

Figure 3. The elapsed times for IPUMS

Figure 4 shows the elapsed time with fourteen curves for **BOOK**. The seven curves *it*1 to *it*100 are (almost) the same, and are related to queries, but not to N (N = 1, 2, …, 100), which are between 34ms and 346ms. Curves *dt*1 to *dt*100 show the average elapsed times for accessing database to retrieve tuples. The result sets are different for various dis-N queries. The larger N means the longer the elapsed time. The DB-times are less than 20ms for $1 \le N \le 3$. If $10 \le N \le 100$, the DB-times are between 50 and 300ms.

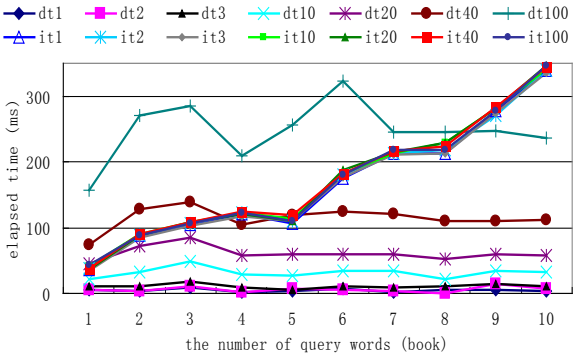

Figure 4. The elapsed times for BOOK

5.2. Precision

It is difficult to confirm whether one of tuples retrieved matches a query semantically by a computer, and it is too big a job to recognize semantic match manually for a large dataset [6]. Therefore, the traditional *recall* for evaluating IR systems is not suitable for measuring semantic match when the dataset is large, and then we

report only *precision*. Figures 5 and 6 illustrate the average *precision* for **IPUMS** and **BOOK** respectively.

We can see that a smaller N has a larger precision; therefore, a smaller N indicates more matching tuples appear in its dis-N results. The precisions for **IPUMS** are 1, 0.93, 0.90, 0.73, 0.62, 0.52, and 0.44 for N = 1, 2, 3, 10, 20, 40, and 100, respectively. The precisions for **BOOK** are 0.93, 0.86, 0.78, 0.63, 0.55, 0.45, and 0.34 for N = 1, 2, 3, 10, 20, 40, and 100, respectively.

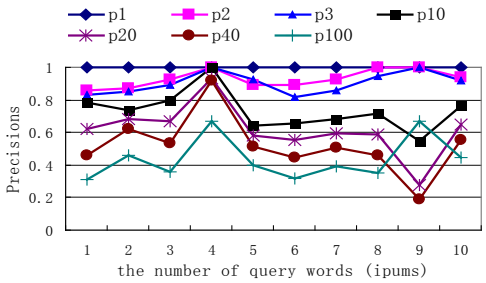

Figure 5. Precisions for IPUMS

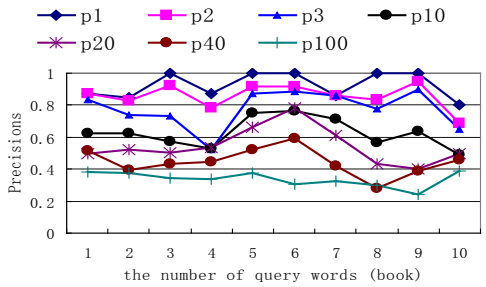

Figure 6. Precisions for BOOK

6. Conclusions

We proposed a new method to evaluate relational ranking queries that reference both text attributes and numeric attributes. The method builds a ranking function combining the semantic and numeric distances, and creates an index based on WordNet to expand the tuple words semantically for a text attribute and on the information of numeric attributes. Thus, the results for a query are retrieved by the index and a simple SQL selection statement for the natural join of relations, and ranked according to the ranking function. We conducted extensive experiments to measure the performance of this new technique using two real datasets. The results of experiments demonstrated that our strategy is efficient and effective.

7. Acknowledgements

This work is supported in part by NSFC (61170039) and the NSF of Hebei Province (F2012201006). The authors would also like to express their gratitude to Professor Weiyi Meng for providing many helpful suggestions.

REFERENCES

[1] M. Carey, and D. Kossmann, "On saying 'enough already!' in SQL," *SIGMOD 1997 Proceedings ACM international conference on management of data*, Vol. 26 No. 2, 1997, pp. 219-230.

[2] I. F Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Computing Surveys*, Vol. 40, No. 4, 2008, Article 11.

[3] L. Zhu, W. Meng, C. Liu, W. Yang, and D. Liu, "Processing top-N relational queries by learning," *Journal of Intelligent Information Systems*. Vol.34, No.1, 2010, pp.21-55, doi:10.1007/s10844-009-0078-7.

[4] IPUMS Census Database, "ipums.la.99.gz", 1999, http://kdd.ics.uci.edu/databases/ipums/ipums.html

[5] T. Pedersen, "*WordNet::Similarity*," 2008, http://www.d.umn.edu/~tpederse/similarity.html

[6] L. Zhu, Q. Ma, C. Liu, G. Mao and W. Yang. "Semantic-distance based evaluation of ranking queries over relational databases," *Journal of Intelligent Information Systems*, Vol. 35, No. 3, 2010, pp. 415-445.

[7] D. Buscaldi, P. Rosso, and A. E. Sanchis, "A wordnet-based query expansion method for geographical information retrieval," *Working Notes for the CLEF Workshop*, Vienna Austria, 2005.

[8] S. Das, E. Chong, G. I. Eadon, and J. Srinivasan, "Supporting ontology-based semantic matching in RDBMS," *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB'04)*, Toronto, Canada, 2004, pp. 1054-1065.

[9] J. Zhang, Z. Peng, S. Wang and H. Niehang, "Si-SEEKER: Ontology-based semantic search over databases," *Knowledge Science, Engineering and Management*, Guilin, China, Vol. 4092, 2006, pp. 599-611.

[10] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-style keyword search over relational databases," *In Proceedings of 29th International Conference on Very Large Data Bases (VLDB'03)*, Berlin, Germany, 2003, pp. 850-861.

[11] F. Liu, C. Yu, W. Meng and A. Chowdhury, "Effective Keyword Search in Relational Database," *26th ACM SIGMOD/PODS international Conference on Management of Data/Principle of Database Systems*, Chicago, Illinois, USA, 2006, pp. 563-574.

[12] L. Zhu, Y. Zhu, and Q. Ma, "Chinese Keyword Search over Relational Databases," *World Conference on Science and Engineering (WCSE)*, Wuhan, China, Vol. 1, 2010, pp. 217 – 220.

