Effective Keyword Search over Relational Databases Considering keywords proximity and keywords N-grams

Sina Fakhraee

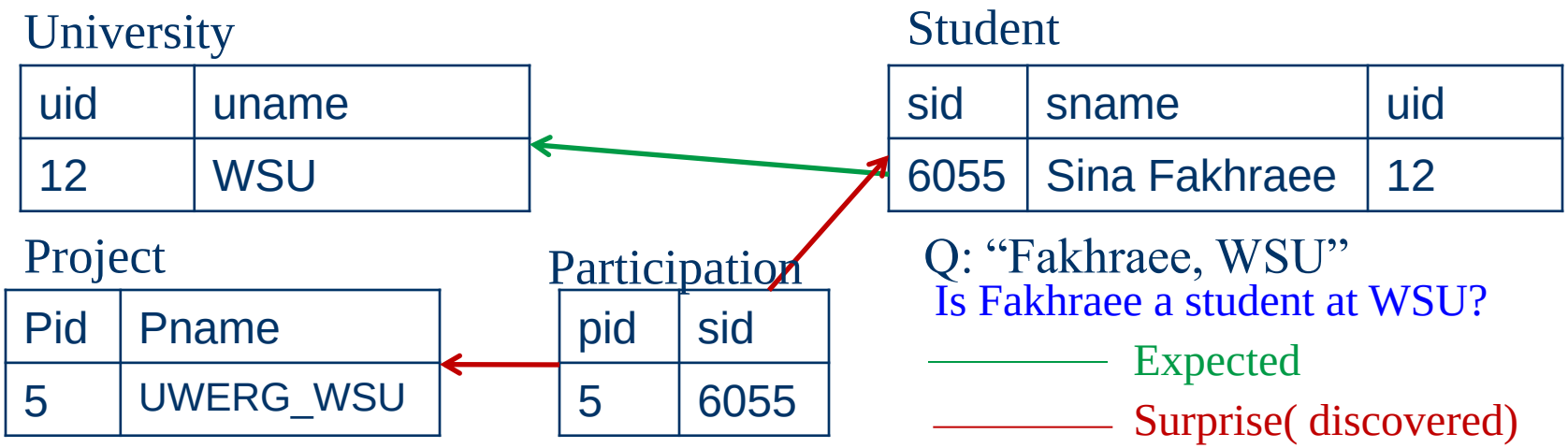
Farshad Fotouhi

WAYNE STATE

College of Engineering

Supporting Keyword-based search over Relational Databases:

- Advantages:
  - Easy to use
    - No need to know SQL and DB schme
  - discovering interesting/unexpected results



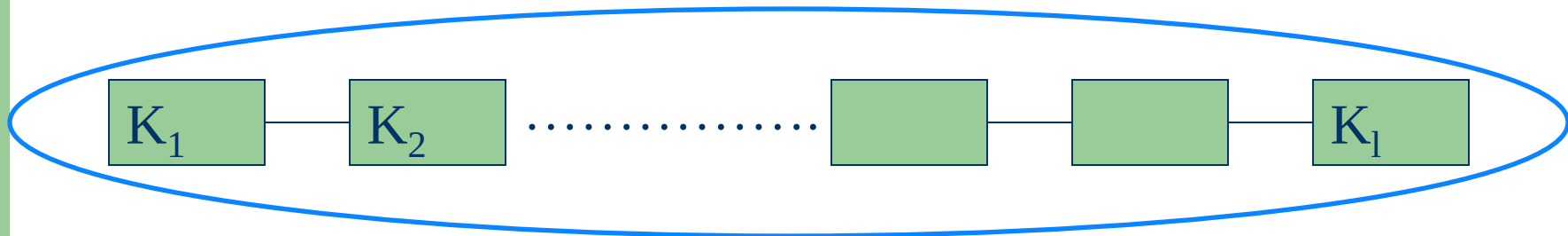
Supporting Keyword-based search over Relational Databases:

- Challenges (Research issues)
  - Search effectiveness

    Why is it difficult to find the most relevant results?
    - Keyword queries are ambiguous:
      - Structural ambiguity
      - Keyword ambiguity
  - Search efficiency
    - Complexity of data and its schema
      - Millions of nodes/tuples
      - Cyclic / complex schema

Problem Description

- **Data**
  - **Relational Databases (graph)**
- **Input**
  - **Query Q = <k_1, k_2, ..., k_l>**
- **Output**
  - **A collection of nodes collectively relevant to Q**



Basic Concepts and Definitions

- **Schema graph:** directed graph denoted by *G(V,E)*
  - *V* represents the set of nodes (relations)
  - *E* are the edges (primary-foreign key relationships among the relations.
- **Tuple**: a single record in the database
- **Tuple tree:** denoted by *T*
  - Also called inter-connected tuples
  - tree of tuples joined on their primary-foreign relationship across DB
- **Keyword Query:** denoted by *Q*
  - set of keywords K_1, K_2,…, K_n entered by the user.

Basic Concepts and Definitions cont'd

- **Query result:** denoted by *R*
  - set of *tuple trees T's*
  - each *tuple tree T* must contain each k_i in at least one of its tuple t_i.
  - **Totality**: all the keywords are contained in *T.*
  - **Minimality**: no tuple can be removed from *T* and still be total.
- **Master Index:** Master index is an inverted list that relates each keyword that appears in the database with a list of locations in the database which are recorded as row-column pairs.

Basic Concepts and Definitions cont'd

- **Tuple set:** denoted by R_i^k (tuple set of R_i with respect to *k)*
  - R_i is a relation
  - *K* is a subset of keyword query *Q*
  - R_i^k is a set of tuples of R_i that only contain keywords in *K* and not any other keywords.
- **Candidate Network:** denoted by *j*(*v*,*e*)
  - also called join tree
  - where $e \in E$ and $v \in V$
  - It is a sub-tree of *G* such that each individual leaf node (relation) contains at least one of the keywords from *Q* and all the leaf nodes together contain all the keywords

Example to illustrate the basic concepts

Schema graph G Tuple Tuple tree Tuple set : M^{Face}

Join tree with respect to Q="Travolta Woo"

Table: Actor(A)

aID	actor
1	J.Travolta
2	M. Ryan
3	L. Bai
4	……

Table: Director(D)

dID	director
1	J.Woo
2	J. Cameron
3	B. Bay
4	……..

Table: Movie(M)

mID	aID	dID	title
1	1	1	Face/Off
2	3	3	Face
3	……	……	……
4	……	….	…..

More tables

N	N
N	N

……

More tables

N	N
N	N

…..

Query Result Generation's Algorithm

- Three modules involved:
  1. Master index:
     - Inputs: keyword query *Q*
     - Outputs: a set of tuple sets for each relation with respect to each keyword.
  2. Candidate network (join tree) generator :
     - Inputs: keyword query *Q*, set of tuple sets, schema graph *G*
     - Outputs: the set of candidate networks (join tress).
  3. Execution engine:
     - Inputs : the set of candidate networks
     - Outputs: the set of answers to *Q* by executing the actual SQL statements corresponding to each candidate network.

Query Result Generation's Algorithm cont'd

- The key component in the pipeline is the candidate network generator
- candidate network generator must generate join trees satisfying 2 criteria:
  1) Totality: $\forall\, k \in Q\,, \exists\, R_i^K \in J \mid k \in K$ (condition 1)
  2) Minimality: $\nexists\, R_i^F \in J \mid R_i^F\ is\ a\ leaf\ node$ (condition 2)

Query Result Generation's Algorithm cont'd

Algorithm1: Candidate Network (Join-Tree) Generation Algorithm
Input: Keyword query *Q*, set of free and non-free tuple sets with respect to *Q* and the schema graph of the relational database
Output: Set of candidate networks, *J-Set*
1: *J-Set* = ∅ // set of candidate networks
2: $queue\ q$ // enqueue all the free and non-free tuple sets
3: *For each* $R_i^{KorF}\ in\ q$; $q\ is\ not\ empty$; *J = deque q*
4: *For each* $R_j^{KorF}\ in\ q$; R_j *is adjacent to* R_i ;*Nod e= deque q*
5: *If* (*J* is not Total && addition of *Node* to *J* does not violate conditions 3)
6: *J = J + Node*
7: *If* (*J* is Total) // (condition 1)
8: *J-Set = J-Set* ∪ *J*
9: *endif*
10: *endif*
11: *end for*
12: *end for*

Background in Ranking Search results

- Early works ranked answer tuple tree *T* based on the size of *T*:

$$score(T, Q) = \frac{1}{size(T)}$$

- Reasoning behind this approach:
  - the closer the keywords are to each other the more relevant the answer tuple tree is to the keyword query *Q*.
- Shortcoming:
  - This is only effective if each tuple contains only one distinct keyword.

Background in Ranking cont'd

- Modern relational databases have incorporated the state-of-the-art Information Retrieval (IR)relevance ranking functionality for individual text attributes.
- This feature is exploited by later works to define their ranking function:

$$Score(d,Q) = \sum_{k \in Q \cap d} \frac{1 + \ln(1 + \ln(tf))}{(1-s) + s\frac{dl}{avdl}} . ln \frac{N}{df}$$

Background in Ranking cont'd

$$Score(d,Q) = \sum_{k \in Q \cap d} \frac{1 + \ln(1 + \ln(tf))}{(1-s) + s\frac{dl}{avdl}} . ln \frac{N}{df}$$

- d is a single text attribute ,
- *k* is a single keyword in *Q*,
- *tf* is the term frequency of *k* in the value of text attribute *d*,
- *df* is the document frequency for keyword *k* which is the number of tuples in *d*'s relation with *k* appearing in *d*'s value,
- *dl* is the document length which is the number of characters of the value of *d*,
- *avdl* is the average length of the value of text attribute *d*,
- *N* is the number of tuples in *d's* relation and
- *s* is a constant number of value 0.2 usually

Background in Ranking cont'd

- Final ranking function:

$$Score(T, Q) = Combine\big(Score(D, Q), size(T)\big)$$
$$= \frac{\sum_{d_i \in D} Score(d_i, Q)}{size(T)}$$

  - *D* is set of all the text attributes of *T (search result)*.

Our contribution

- In this work we have identified two other important factors that can further improve the search effectiveness when incorporated to the IR relevance ranking strategy.(Final ranking function shown)
  - **Keyword proximity:** which is the overall distance of the keywords from one another in the value of the target text attribute
  - **keyword Quadgrams:** We computed keyword Quadgrams of the keywords in both the query itself and in the text attributes.
    E.g. Fakhraee, -fakh, akhr, raee, hraee

Keyword Proximity

- **Minimum pair distance proximity:** is the smallest distance of all the pairs of distinct matched query keywords in the target document
  - denoted by *MinDist*(*Q*,*D*)
  - *D* is the target document
  - Q is the query.
- The formal definition is:

$$MinDist(Q, D) = min_{k1,k2 \in Q \cap D}\{Dist(k1, k2)\}$$

  - Where, $Dist(k1, k2)$ is the length of the shortest segment between *k1* and *k2* in *D*

Incorporation of Keyword Proximity into the Relevance Ranking Function

- Proximity values cannot simply be added to ranking function since these two quantities are not comparable.
- Proximity function, *MinDist*, should be transformed to a function that produces values that are comparable with ranking function. (τ = Tau)

$$\tau(Q, D) = \gamma(MinDist(Q, D))$$

- Where γ is the transformation function and $\tau(Q, D)$ is the new transformed function. (γ= gamma)

Incorporation of Keyword Proximity into the Relevance Ranking Function cont's

- Two criteria must be met by the new transformed function:

1. $\tau(Q, D)$ should positively impact the relevancy of the document to the query. (i.e. the smaller the *MinDist* the larger $\tau(Q, D)$).
2. The effect of *MinDist* on $\boldsymbol{\tau}$ should drop quickly as the distance gets smaller past some point (to not to keep compensating that document) and its effect should become constant as the distance becomes larger beyond some point (to not to keep penalizing the document with larger distance between keywords).

Incorporation of Keyword Proximity into the Relevance Ranking Function cont's

- These two constraints lead to the following definition:

$$\tau(Q,D) = \log(\alpha + \exp(-MinDist(Q,D)))$$

- We have adopted this definition for τ as our adjustment factor to be added directly to the final ranking function shown before. (*D* is replaced by the target text attribute in the *DB)*

Keyword N-Grams

- Some users might misspell query keywords
- Some users might only know the partial spellings of the keyword(s)
- More importantly keywords might have been misspelled in the target text attributes' values.

To demonstrate these problems, consider the following three examples:

Keyword N-Grams cont'd

- Example 1) Assume a user is searching for action movies featuring actor *John Travolta* and is not sure about the correct spelling of the name of the actor. Therefore, he might perform his search by typing for example keyword sequences such as "*Actions Travelta*" or "*Actions Traveltha*" instead of "*Actions Travolta*".

How does this impact the search?

➢This could result in the failure of finding the intended inter-connected tuples depending on which SQL predicate (such as *LIKE*, *CONTAINS* or *FREETEXT*) was used to implement the search system.

Keyword N-Grams cont'd

- Example 2) Assume a user is searching for thriller movies featuring a German actress *Martina Gedeck*. Also assume that the targeted movie database has inconsistencies in how the actress name has been spelled (e.g. *Gedack*, *Gedek* , etc.),

How does this impact the search effectiveness?

➢this could negatively impact the effectiveness of the keyword search because the *tf* and *idf* will not be accurate.

Keyword N-Grams cont'd

- Example 3) Similar to the second example, if the values of a target text attribute contains different variations of the same verb or noun (e.g. verb "category" and its variations such as "categories", "categorization" , etc).

How does this impact the search effectiveness?

➢this could also negatively impact the effectiveness of the keyword search because the *tf* and *idf* factors of the ranking function will not be accurate due to possible mismatching between query keywords and the keywords in the text attributes' values.

Keyword N-Grams cont'd

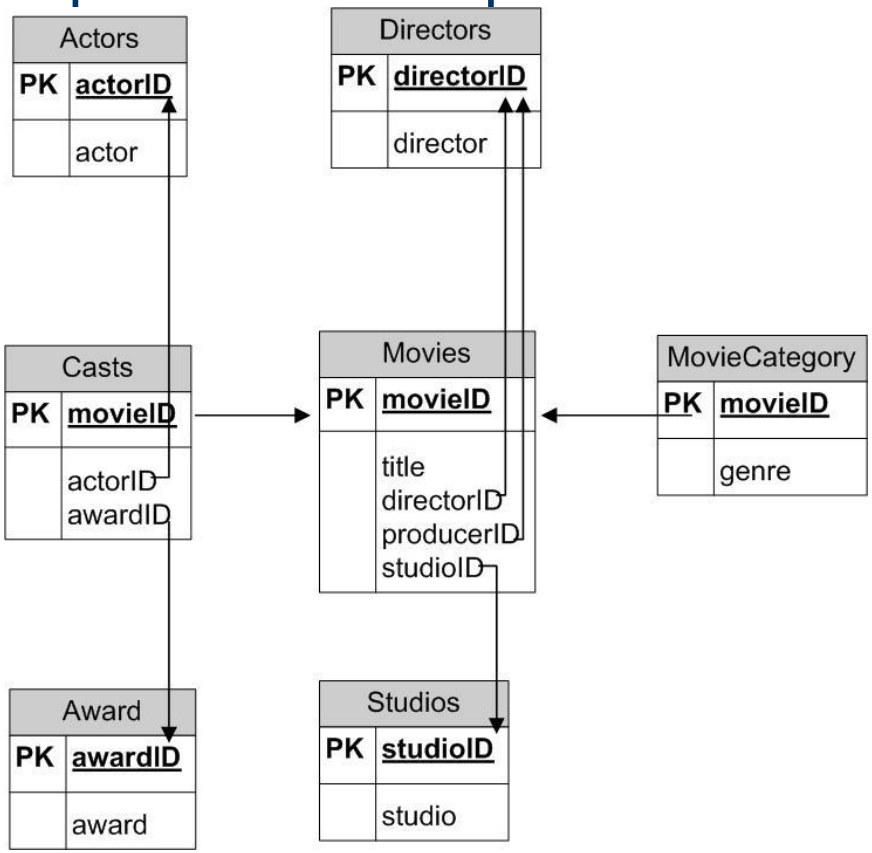
- To address these issues we have computed the keyword Quadgrams in both:
  - the values of the target text attributes and
  - In the query itself
- We incorporate the Quadgrams to ranking function by
  - updating the *tf* and *idf* not only when we encounter the exact search terms but also when we encounter the Quadgrams.

The final ranking algorithm

Algorithm2: Relevance Ranking Algorithm
Input: Keyword query *Q*, a set of answer tuple trees *T'*s
Output: A ranked set of answer tuple tree.
1: *T-Set //* set of answer tuple trees *T'*s
2: *T-Ranked-Set //* a priority queue to store the set of
3: // ranked answer tuple trees *T'*s
4: *Q* // query keywords
5: *k_Quadgrams* //a set of all the quadgrams for keyword *k*
6: *For each* k_i in *Q*
7: computeQuadgrams(k_i)
8: k_i*_Quadgrams.update*
9: *end for*
10: *For each* k_i in each text_attribute
11: computeQuadgrams(k_i)
12: k_i*_Quadgrams.update*
13: *end for*
14: *For each T* in *T-Set*
15: *T-Ranked-Set* .push(*T, Score'*(*T*,*Q*)) // *Score'*(*T*,*Q*) is
16: // the modified version of Formula 3 after applying
17: // Quadgrams and $\tau(Q, d)$ to Formula 2.
18: *end for*
18: *Return T-Ranked-Set*

Experimental Results

- We used the IMDB dataset to perform our experiments.
  - *#Actors 2000*
  - *#Directors 1200*
  - *#Movies 4000*
  - *#PlotSummary 4000*
- Database contains misspellings for the name of some of the actors/directors/characters/titles and in the plot summary of the movies.



Experimental Results cont'd

- We created two types of queries:
  1. *Type I* queries, targeting both short and long text attributes.
  2. *Type II* queries, only targeting short text attributes

	Type I queries		Type II queries
1	Summary	7	actor, genre
2	Summary, director	8	actor, director
3	Summary, actor	9	director, genre
4	Summary, genre	10	actor, director, genre
5	Summary, actor, genre	11	actor, character
6	Summary, character		

Experimental Results cont'd

- We formulated 50 queries, 25 per each type.
- we used two measures:
  - 1) Number of top-1 search results that are relevant denoted by *#Rel* .
    - (We chose this metric to evaluate *type I* queries since the user's primarily intention is to find a single movie.)
  - 2) precision/recall curve
    - We chose this metric to evaluate *type II* queries since the user's primarily intention is to find a set of relevant movies.

Experimental Results cont'd

- We identified the relevant answers in our database as follows: (i.e. pooled relevance judgment)
  - We ran all four algorithms for each query
  - 1) *BA*: base algorithm
  - *2) BA+KP*: base algorithm + keywords proximity
  - 3) *BA+KQ* = base algorithm + keywords quadgran
  - 4) *BA+KP+KQ* = base algorithm + keywords proximity + keyword quadgrams
  - we merged their *top20* results.
  - We then manually judged and selected relevant results for each query out of the 80 candidate results.

Experimental Results cont'd

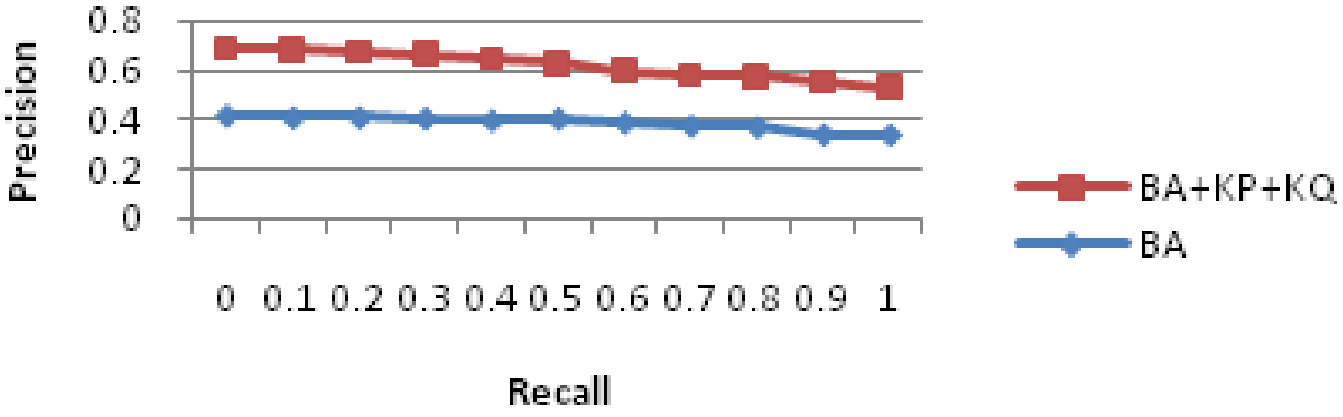
- To evaluate the effectiveness and impact of keyword proximity and keyword quadgrams:
- for *type II* queries:
  - 12 out of 25 submitted queries contained misspellings.
- for *type I* queries:
  - 12 out of 25 submitted queries contained misspellings and phrases with different keywords' variations.(e.g. category vs. categories)

Experimental Results cont'd

- Table below shows top-1 search results for *type I* queries for each algorithm

	BA	*BA+KP*	*BA+KQ*	*BA+KP+KQ*
#Rel	8	12	13	16

- Figure below shows the 11-point precision/recall graph for *type II* queries for *BA* and *BA+KP+KQ* algorithms. (#Rel=# of top1 results that are relvevent)



Conclusion

- We can see from *Table III* that *BA+KP+KQ* algorithm outperformed the base algorithm *BA* for *type I* queries. We observed that many of the relevant answers to the queries targeting the *plot summary*, which contained misspelled keywords or keywords which were located apart from one another, were not in the top-1 search results for *BA*, and in fact they were not even in the top-10 results. We can also see from Figure 4 that *BA+KP+KQ* algorithm outperformed the base algorithm *BA* for type II queries as well.

Thank you!