Weight-Adjustable Ranking for Keyword Search in Relational Databases

Chichang Jou[1*] and Sian Lun Lau[2]

[1]Department of Information Management, Tamkang University, Taiwan
[2]Department of Computing and Information Systems, Sunway University, Malaysia
cjou@mail.tku.edu.tw, sianlunl@sunway.edu.my

Abstract. Huge volumes of invaluable information are hidden behind web relational databases. They could not be extracted by search engines. The problem is especially severe for long text data, for example: book reviews, company descriptions, and product specifications. Many researches have investigated to integrate information retrieval and database indexing technologies to provide keyword search functionality for these useful contents. Due to diversifying data relationships in application domains and miscellaneous personal preferences, current ranking results of related researches do not satisfy user requirements. We design and implement a Weight-Adjustable Ranking for Keyword Search (WARKS) system to address the issue. Mean average precision (MAP) and mean rank reciprocal difference (MRRD) are proposed as measurements of ranking effectiveness. We use an integrated international trade show database as our experimental domain. User study demonstrates that WARKS performs better than previous practices.

Keywords: Keyword Search, Information Retrieval, Ranking, Mean Average Precision, Rank Reciprocal Difference.

1 Introduction

Along with the rapid accumulation of internet contents, search engines have been very important tools in retrieving and collecting information. However, lots of invaluable data are still hidden behind web relational databases. The problem is especially severe for data in long text formats, such as book reviews, company descriptions, and product specifications. They could not be indexed by traditional numeric or short-term indexing of relational databases. Many researchers have tackled the problem by integrating information retrieval and database indexing to provide keyword search functionality for these useful contents.

Due to diversifying data relationships in application domains and miscellaneous personal preferences, current ranking results of related researches do not satisfy user requirements. Take the domain of international business trading as an example. All trade show web sites store detailed information about participating companies and their products in web relational databases. Their company table includes long text columns for company descriptions, and their product table includes long text columns for product specifications. International traders often need to integrate information about which

companies attended which trade shows with which products, especially when they are not certain about the product names. To decide which company to enhance business cooperation, they also need to collect accumulated information across web sites about the participating companies, like frequencies, experiences, product categories, exhibition lot sizes, etc. The requirements of personalized keyword search functionality to retrieve company or product information for trade show web sites are imminent.

We extend ranking mechanisms of previous researches to design and implement a Weight-Adjustable Ranking for Keyword Search (WARKS) system to meet user requirements. WARKS augments adjustable column weights and tuple weights to rank the retrieved results. Measurements of mean average precision (MAP) and mean rank reciprocal difference (MRRD) are proposed as ranking effectiveness indicators. We use an integrated international trade show database as our experimental domain. User study demonstrates that WARKS performs better than previous practices.

2 Related Work

Bergamaschi et al. [4] surveyed keyword search for relational databases and identified challenging tasks in this research area. Simitsis et al. [14] categorized processing principles for keyword search in relational databases into the following two approaches: based on database schema and based on tuples.

2.1 Based on database schema

This approach treated database schema as a graph, where each node represents a table, and each edge represents a referential relationship between tables. The following are related researches in this approach:

Discover [8] utilized the join mechanism of relational databases. The system first retrieved those joinable candidate tuple networks (CTN), and then evaluated their execution plans. It used greedy algorithm to restrict number of retrieved CTNs. DiscoverII [9] adopted ranking technology to boost efficiency. It first performed full text retrieving. The obtained tuples were associated by foreign keys to form CTNs, which were then transformed into SQL syntaxes. The top ranked results were returned to the user.

DBXplorer[1] performed breadth-first traversal for all tuples in the tuple graph, and built the SQL syntaxes to obtain related tuples. The system considered not only size of the result sets, but also relevance of the query results with respect to the keywords.

Keymantic [3] parsed keyword phrases from knowledge bases composed by database schema, data types, data dictionary mapping, and ontology. It listed all possible parsing trees for a query, and let the user select the one that was closest to their intention. The system then transformed the selected parsing into relevant SQL syntaxes.

2.2 Based on tuples

This approach treated database contents as a directed graph, where each node represents a tuple, and each edge represents a reference between two tuples. Under this principle, these systems directly built tuple trees that satisfy all requirements (AND semantics). The following are related researches in this approach:

BANKS [6] used backward expanding search for graph traversal to retrieve all keyword-matched tuples in the tuple graph. The search started from each keyword-matched tuple, and then followed the backward direction of an edge. However, for tuples with many keyword matchings, it had a poor search efficiency. BANKSII[11] proposed bi-directional search so that keyword search could start from the root in a tuple graph. This mechanism improved the efficiency shortcoming of BANKS for tuples with many keyword matchings.

Liu et al. [12] proposed a ranking strategy that performed normalizations of tuple tree size, document length, term frequency, and inverse document frequency for candidate tuple trees. Schema terms were given higher scores.

Objectrank [2] applied authority-based ranking. They built a directed graph, where each node was tagged with a node type to represent the tables they were in. Contents of a node is represented by a set of keywords. Weights of all nodes, representing their authorities, were iteratively calculated. The system could support both AND-semantics and OR-semantics for the query keywords.

2.3 Ranking evaluation and recent developments

Coffman et al. [7] proposed a framework to quantitatively evaluate effectiveness of keyword search and their ranking mechanism. Bergamaschi et al. [5] tried to induct the user intentions in choosing keywords. These intentions are then integrated with data representations in the databases to retrieve results satisfying user requirements.

Jabeur et al. [10] proposed user-centric product search for e-commerce sites by taking user engagement into account. In addition to product features, they included social interactions, namely "like" and "share" tags, in ranking the retrieved results.

Liu et al. [13] extended keyword search to the temporal graphs for users to specify optional predicates and ranking functions related to timestamps. Three types of ranking functions are supported: relevance, time, and duration. They extended Dijkstra's shortest path algorithm to find the best paths between two nodes in each time instant with respect to a ranking function.

Zhu et al. [15] tackled the problem that keyword search results are biased by duplicate tuples that refer to the same real-world entity. They designed a clustering algorithm using the divide-and-conquer mechanism to reduce duplicates in the results.

3 Design and implementation of WARKS

3.1 Data Model of WARKS

WARKS adopts the data model proposed by Hristidis et al. [9] and Liu et al. [12]:

- Database schema diagram (abbreviated as *DSD*): Suppose there are n tables in the database. A node is designated for each table. If there exists a relationship from table R_i to table R_j, then a directed edge from R_i to R_j is added. Their association degree (1-to-1 or 1-to-many) is augmented to the edge.
- Tuple tree (abbreviated as *T*): Given a *DSD*, each node in the tree represents a tuple (aka record). A tuple tree is constructed by several related tuples. Suppose $<R_i, R_j>$

is an edge in a *DSD*, if tuple $t_i \in R_i$, $t_j \in R_j$, and (t_i *join* t_j)$\in$(R_i *join* R_j), then the directed edge $<t_i, t_j>$ is added.

- Query (abbreviated as *Q*): WARKS deletes stop words in query terms, and performs stemming on the remaining terms. The remaining terms are treated as a set.

In the integrated trade show database, the following tables are constructed: TradeShow, Company, and Product. Its *DSD* is displayed in Figure 1.

The bridge tables CompanyProduct, TradeShowCompany, and TradeShowProduct demonstrate foreign key existence in the participating tables. One functionality in WARKS is to use keywords to query the following attributes: the company description of the Company table, the product name of the Product table, and the product specification of the Product table. The product name attribute will be matched using traditional indexing, while the other two attributes will be matched using full text indexing.

WARKS provides a keyword input text field for searching related companies and their products. We use a company tuple as root of the tuple tree, and build tuple trees as displayed in Figure 2.

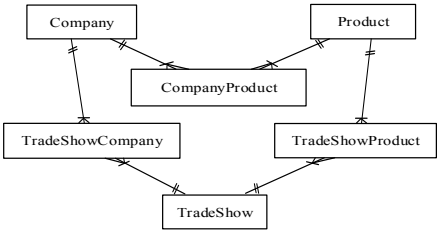

Fig. 1. Example database schema diagram

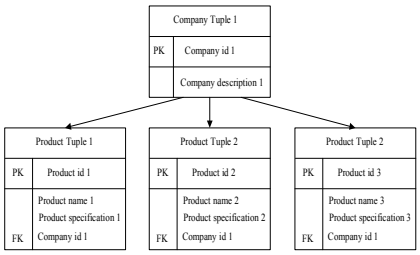

Fig. 2. Example tuple tree

3.2 Ranking Mechanism of Previous Works

This section introduces the ranking mechanism proposed by Hristidis et al. [9] and Liu et al. [12]. Suppose each tuple is represented as one document *D*, and the document collection in a tuple tree *T* is the set $\{D_1, D_2, \ldots, D_m\}$. For a query phrase *Q*, the ranking for a returned *T* is based on the similarity measure of *Q* and *T*, which is computed by summing the products of the weights of terms in *Q* and those in *T*.

$$Sim(Q,T) = \sum_{k \in Q \cup T} weight(k,Q) * weight(k,T) \quad (1)$$

In equation (1), *weight(k, Q)* is the term frequency of *k* in *Q*, and *weight(k, T)* is computed through the steps expressed in equations (2) to (7):

$$weight(k,D) = \frac{ntf(k,D) * idf^g(k)}{ndl(k,D) * Nsize(T)} \quad (2)$$

Equation (2) is the weight of each term k in document *D* of *T*. To avoid ranking bias, this weight is computed through the following four normalization steps:

- *Tuple tree size normalization (Nsize(T))*: In equation (3), *size(T)* is the number of tuples in *T*, *avgsize* is the average tuple tree size for all tuple trees. Following previous practices, parameter *s* in equation (3) is set as 0.2.

$$Nsize(T) = (1-s) * s * \frac{size(T)}{avgsize} \tag{3}$$

- *Document length normalization (ndl(k, D))*: In equation (4), document length *dl(k,D)* is to the occurrence number of *k* in *D*, and *avgdl* is the average number of words for all retrieved documents. Parameter *s* in equation (4) is also set as 0.2.

$$ndl(k,D) = \begin{cases} \left((1-s) + s * \frac{dl(k,D)}{avgdl}\right) * (1 + \ln(avgdl)) & k \in D \\ 0 & k \notin D \end{cases} \tag{4}$$

- *Term frequency normalization (ntf(k, D))*: It is against intuition to have the similarity score linearly proportional to occurrence numbers of a term in the document. Equation (5) is to normalize the number of occurrences for *k* in *D*:

$$ntf(k,D) = 1 + \ln\left(1 + \ln\left(tf(k,D)\right)\right) \tag{5}$$

- *Inverse document frequency normalization (idf^g(k))*: Previously, the normalization computation had been restricted to the local data returned by a search. This was extended to the *global* scale. In equation (6), *df^g(k)* is the number of documents where *k* appears. *N^g* is the number of documents in the corpus.

$$idf^g(k) = \ln \frac{N^g}{df^g(k)+1} \tag{6}$$

- *The weight of k in T (weight(k, T))*: In equation (7), $max_{D \in T}\, weight(k,D)$ is the maximum occurrence number of *k* for all *D* in *T*, and $\sum_{D \in T} weight(k,D)$ is the sum of weights of *k* for all *D* in *T*:

$$weight(k,T) = \max_{D \in T} weight(k,D) * \left\{1 + ln\left(1 + \frac{\sum_{D \in T} weight(k,D)}{\max_{D \in T} weight(k,D)}\right)\right\} \tag{7}$$

The ranking mechanism of previous work has the following shortcomings:

1. Contents of all columns in a tuple were treated equally.
2. All tuples in the database were treated equally.

3.3 Design of WARKS

We propose to increase the weights of keywords in important columns and tuples. Use the trade show database as an example, to increase the weights of keywords in the "product name" column, we could emphasize the *tf(k,D)* in the term frequency normalization (equation 5).

We propose the adjustable ranking mechanism by allowing the user to set up the following two parameters:

1. The weight α for number of occurrences of *k* in a specific column: To indicate the importance, the value of α must be greater than or equal to 1. WARKS differentiates the importance of individual columns by multiplying occurrence numbers of terms

in column *i* by the weight α_i. Suppose there are *l* columns in document D. In equation (8), n_{ki} denotes *k*'s occurrence numbers in column *i*.

$$tf(k,D) = \frac{\sum_{i \in \{1,\ldots,l\}}(n_{k_i} * \alpha_i)}{\sum_{t \in D}(\sum_{i \in \{1,\ldots,l\}}(n_{t_i} * \alpha_i))} \quad (8)$$

2. The weight β for tuple types: To differentiate the importance of the tuples, WARKS allows the user to adjust the weight of *k* in a document (equation (9)). According to the tables that a tuple belongs to, WARKS first categorizes tuples, and then assigns weight β_i to the *i*-th tuple type. To increase the variation range, the value of β_i will be set as 2^n, where *n* is between -10 and +10.

$$weight(k,D) = \begin{cases} \frac{ntf(k,D) * idf^{\vartheta}(k,D)}{ndl(k,D) * Nsize(T)} * \beta_i & D \text{ is of the } i\text{-th tuple type} \end{cases} \quad (9)$$

3.4 Implementation of WARKS

WARKS is implemented using PHP 5.3.5 in a Microsoft Server 2012 server using Intel Xeon CPU E5620 2.13GHz CPU with 12 GB memories. We crawled trade show, company and product data from 10 web sites. They are integrated into a MySQL 5.6 database. We utilized full text indexing of MySQL in creating the related tables to assist answering keyword query. Totally there are 6,630 companies and 23,077 products.

WARKS utilized mySQL's "Match Against" function The Match function supports natural language processing. The Against function supports search using the keyword terms. WARKS applies the Match Against syntax to the company description and the product specification columns in the Company and Product tables, respectively.

The term "notebook" is used as a keyword search example to illustrate the adjusted normalization in WARKS. WARKS retrieves 29 company records, and 122 product records for "notebook". Using traditional SQL indexing of the product name column, WARKS retrieves 54 product records. After eliminating duplicated records, 133 product records under 29 companies are returned.

The statistics regarding the query term "notebook" in the Company and Product records are displayed in Table 1. The tuple tree for this company is built by the following foreign keys: C430→CP1, P1916→CP1, C430→CP2, P1917→CP2, where C430 is the company id, P1916 and P1917 are the product id's, and CP1 and CP2 are record id's in the CompanyProduct bridge table. Table 2 shows the related numbers in the normalization steps for computing the similarity scores for the keyword "notebook" (*Q*) and the tuple tree (*T*).

Table 1. Statistics of the query term "notebook"

column	no. of tuples	total no. of words	Avg tuple size
company description	29	4,011	138.3103
product specification	133	31,363	235.8158

Table 2. Computed numbers in the normalization steps

		Nsize(T)	*ndl(k,D)*		*ntf(k,D)*		*idf*$^{\alpha}$*(k)*		*weight(k,D)*	
Company	*Tuple id*	*Nsize(T)*	*Company*	*Product*	*Company*	*Product*	*Company*	*Product*	*Company*	*Product*
430	C430	1.406	8.939	0	3.253	0	3.776	0	0.977	0
430	P1916	1.406	0	7.673	0	2.496	0	6.255	0	1.447
430	P1917	1.406	0	8.116	0	2.716	0	6.255	0	1.488

4 Experiments and discussions

4.1 Environment Setup

From a web directories of industrial products, we collected for the product categories "Consumer Electronics, Electronic Components, and Computer Peripherals" 30 keyword terms, for example: "android", "battery", "camera", "charger", etc.

Using the pooling method, we invited 5 domain experts to collect query result sets. For each keyword term, most frequently chosen 10 companies and products were selected for the evaluation reference. Then we invited 30 junior and senior university students majored in Information Management as our testers. If a result was chosen by more than half of the testers, then it is chosen as an objective answer.

In our experiments, users could adjust the column weights α only for the product name column, and the tuple weight β only for the product tuples. By fixing α value, users could choose the best β value that fits their preference.

To evaluate the effectiveness of our proposed mechanism, we use the following two indicators: 11-Point mean average precision (MAP) and mean rank reciprocal difference (MRRD). MAP is used to evaluate the precision rate of the retrieved result, and MRDD is used to evaluate the consistency of the retrieved result:

1. *MAP*: MAP has been used in many competitions, like TREC, NTCIR, and CLEF. We apply interpolation to obtain the average precision of 11 fixed recalls. In MAP, for keyword k, average precision of each result is obtained using equation (10), where R_k is the number of query results matching the evaluation reference, P_{kj} is the precision rate for the j-th correct result, and N_Q is number of tested keywords. In other words, MAP is the mean value of the 30 APs.

$$AP_k(\alpha,\beta) = \frac{\sum_{j=1}^{R_k} P_{k,j}}{R_k} \qquad MAP(\alpha,\beta) = \frac{\sum_{k=1}^{N_Q} AP_k(\alpha,\beta)}{N_Q} \tag{10}$$

2. *MRRD*: For each query term k, from the top 10 results produced by all combinations of α and β, we identify the company (or product) with the best rank $(F_k(\alpha,\beta))$, and that with the worst rank $(L_k(\alpha,\beta))$. $RRD_k(\alpha,\beta)$ is defined in equation (11). If for some α, β, and k, $RRD_k(\alpha,\beta)$ is less than 0, it means under that (α,β) setting, the returned query results for k demonstrated a flip-flop phenomenon. For each combination of (α,β), MRRD is the mean value of the 30 RRDs.

$$RRD_k(\alpha,\beta) = \frac{1}{F_k(\alpha,\beta)} - \frac{1}{L_k(\alpha,\beta)} \qquad MRRD(\alpha,\beta) = \frac{\sum_{k=1}^{N_Q} RRD_k(\alpha,\beta)}{N_Q} \tag{11}$$

We polled the users with the two questionnaires to proceed with our user study

- For the method proposed by Liu et al. [12]: By setting α as 1, and β as 2^0.
- For a combination setting (α,β): We obtain a best combination setting (α,β) for *MAP* and *MRRD* separately.

4.2 Experiment results and discussions

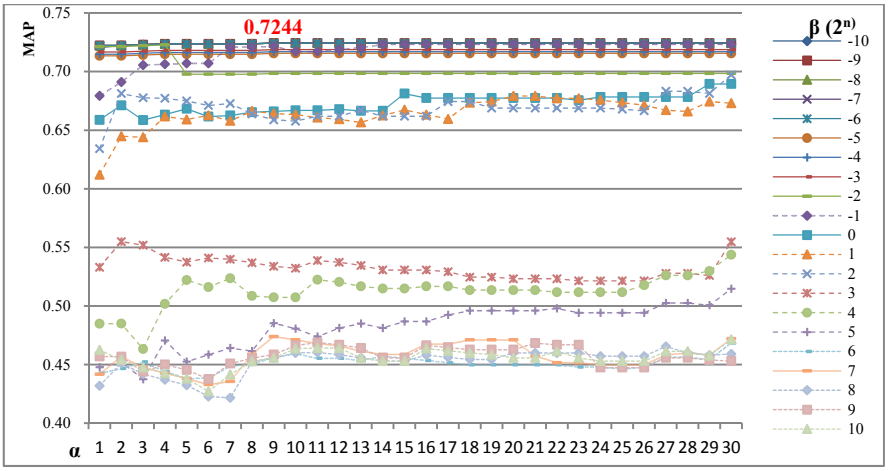

Fig. 3. MAP performance for fixed β

For MAP, we fixed the value of β from the set { 2^{-10}, 2^{-9}, … , 2^9, 2^{10} } first. Then the value of α was set from 1 to 30. For each pair of (α,β), we calculated their MAP value. The result is shown in Figure 3. When β is 2^{-7} and α is 7, the MAP measurement achieves the maximum value 0.7244. With fixed α from 1 to 7, the effect of β on the MAP value is displayed in Figure 4. When β is less than 2^{-7}, the MAP value is almost fixed. The fluctuation periods were when β is from 2^{-2} to 2^4.

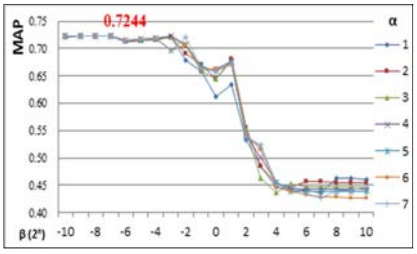

Fig. 4. MAP performance for fixed α

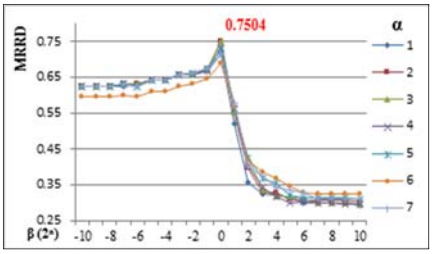

Fig. 6. MRRD performance for fixed α

For MRRD, we fixed the value of βfrom the set {2^{-10}, 2^{-9},…, 2^9, 2^{10} } first. Then the value of α was set from 1 to 30. For each pair of (α,β), we calculated their MRRD value. The result is shown in Figure 5. When β is 2^1, and α is 2, the MRRD measurement achieves the maximum value 0.7504. There is a trend that when α is greater than 11,

the MRRD value becomes very stable for most β values. The line for β equal to 2^2 is a separator. In the above group, the lesser the value of β, the smaller the value of MRRD. In the below group, the more the value of β, the smaller the value of MRRD. With fixed α from 1 to 7, the effect of β on the MRRD value is displayed in Figure 6. When β is 2^1 and α is 2, the MRRD measurement achieves the maximum value 0.7504. In these charts, for a fixed α, when β is less than 2^1, the greater the value of β, the bigger the value of MRRD. For a fixed α, when β is greater than 2^4, the greater the value of β, the smaller the value of MRRD. The fluctuating part is when the value of β is between 2^1 and 2^4.

For user study, we use a fixed pair of α equal to 9 and β equal to 2^{-7}. By checking the average results returned for the 30 keyword terms, we found that using the ranking mechanism proposed by Liu et al. [12], the MAP value is 0.4052, while the MAP value is 0.6471 in WARKS. WARKS has a better performance of 59.70%.

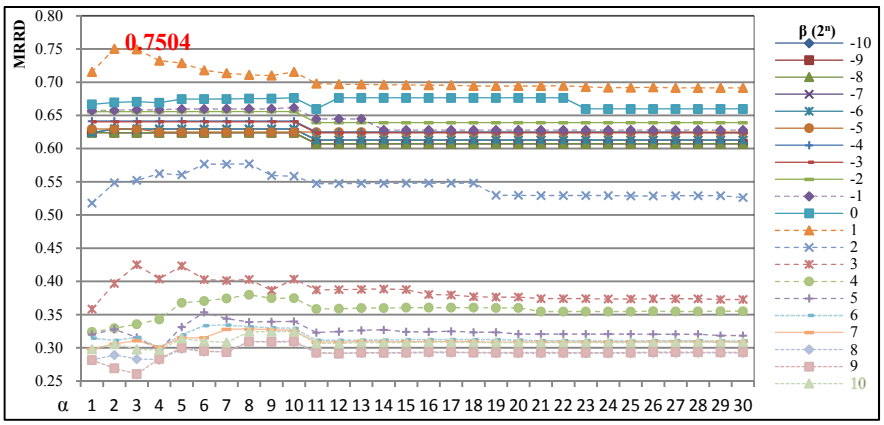

Fig. 5. MRRD performance for fixed β

5 Conclusions

There are many valuable data hidden in web databases, and they could not be retrieved through traditional search engines. We crawled and integrated web trade show databases by collecting company and product records. We provided a keyword search user interface with an option of "company" or "product". We designed and implemented WARKS that provided ranked keyword-matching company (or product) records based on user preferences.

In addition to the four normalization steps in Liu et al. [12], WARKS provided adjustable column weight α and tuple weight β. We used mean average precision (MAP) and mean rank reciprocal difference (MRRD) to evaluate ranking effectiveness of WARKS. Our user study for the case when α is 9 and β is 2^{-7} showed that WARKS provided a better performance of 59.70% improvement than that of Liu et al. [12].

We would like to point out the following future work directions: (1) Extend the research into other domains. (2) Consider the relative position of these keywords. (3)

Increase the weight for frequent input keywords. (4) Incorporate semantic annotation tool for the user.

References

1. Agrawal, S., Chaudhuri, S. & Das, G. (2002). DBXplorer: A system for keyword-based search over relational databases. Proceedings of the 18th IEEE international conference on data engineering (ICDE 2002), p. 5-16.
2. Balmin, A., Hristidis, V. & Papakonstantinou, Y. (2004). Objectrank: Authority-based keyword search in databases. Proceedings of the 30th international conference on Very Large Data Bases (VLDB ’04), p. 564-575.
3. Bergamaschi, S., Domnori, E., Guerra, F., Orsini,M., Lado, R.T. & Velegrakis, Y. (2010). Keymantic: Semantic keyword-based searching in data integration systems. Proceedings of the VLDB Endowment, 3(1-2), p. 1637–1640.
4. Bergamaschi, S., Guerra, F. & Simonini, G. (2014). Keyword Search over Relational Databases: Issues, Approaches and Open Challenges. Lecture Notes in Computer Science 8173, p. 54-73.
5. Bergamaschi, S., Guerra, F., Interlandi, M., Lado, R.T. & Velegrakis, Y. (2016). Combining user and database perspective for solving keyword queries over relational databases. Information Systems, 55, p. 1-19.
6. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S. & Sudarshan, S. (2002). Keyword searching and browsing in databases using BANKS. Proceedings of the 18th IEEE international conference on data engineering (ICDE 2002), p. 431-440.
7. Coffman, J. & Weaver, A. C. (2010). A framework for evaluating database keyword search strategies. Proceedings of the 19th ACM international conference on Information and knowledge management, p. 729-738.
8. Hristidis V. & Papakonstantinou, Y. (2002). DISCOVER: Keyword Search in Relational Databases. Proceedings of VLDB '02, p. 670-681. VLDB Endowment, August 2002.
9. Hristidis V., Gravano, L. & Papakonstantinou, Y. (2003). Efficient IR-style Keyword Search over Relational Databases. Proceedings of VLDB '03, p. 850-861.
10. Jabeur, L.B., Soulier, L., Tamine, L. & Mousset P. (2016). A product feature-based user-centric ranking model for e-commerce search. Lecture Notes in Computer Science, Vol. 9822, p. 174-186.
11. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R. & Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. Proceedings of the 31st international conference on Very Large Data Bases (VLDB ’05), p. 505-516.
12. Liu, F., Yu, C., Meng, W. & Chowdhury, A. (2006). Effective keyword search in relational databases. Proceedings of ACM SIGMOD '06, p. 563-574.
13. Liu, Z., Wang, C. & Chen, Y. (2017). Keyword Search on Temporal Graphs. IEEE Transactions on Knowledge and Data Engineering, 29(8), p. 1667 – 1680.
14. Simitsis, A., Koutrika, G. & Ioannidis, Y.E. (2008). Pr´ecis: from Unstructured Keywords as Queries to Structured Databases as Answers. VLDB Journal, 17(1), p. 117–149.
15. Zhu, L., Du, X., Ma, Q., Meng, W. & Liu, H. (2018). Keyword search with real-time entity resolution in relational databases. Proceedings of the 2018 10th International Conference on Machine Learning and Computing, p. 134-139.