TreeCluster: Clustering Results of Keyword Search over Databases

Zhaohui Peng[1], Jun Zhang[1,2], Shan Wang[1], and Lu Qin[1]

[1] School of Information, Renmin University of China,
Beijing 100872, P.R. China
{pengch, zhangjun11, swang, qinlu}@ruc.edu.cn
[2] Computer Science and Technology College,
Dalian Maritime University, Dalian 116026, P.R. China

Abstract. A critical challenge in keyword search over relational databases (KSORD) is to improve its result presentation to facilitate users' quick browsing through search results. An effective method is to organize the results into clusters. However, traditional clustering method is not applicable to KSORD search results. In this paper, we propose a novel clustering method named TreeCluster. In the first step, we use labels to represent schema information of each result tree and reformulate the clustering problem as a problem of judging whether labeled trees are isomorphic. In the second step, we rank user keywords according to their frequencies in databases, and further partition the large clusters based on keyword nodes. Furthermore, we give each cluster a readable description, and present the description and each result graphically to help users understand the results more easily. Experimental results verify our method's effectiveness and efficiency.

1 Introduction

Based on the full text indexing provided by RDBMS, keyword search over relational databases (KSORD) enables casual users to use keyword queries (a set of keywords) to search relational databases just like searching the Web, without any knowledge of the database schema or any need of writing SQL queries[1, 2]. The recent studies on KSORD can be categorized into two types according to the search mechanism, *schema-graph-based* and *data-graph-based*. The former includes DBXplore[5], DISCOVER[6], IR-Style[7]. The latter can be further classified into two types based on the search results. One is those that return a single tuple as result, e.g. ObjectRank[8]. The other, e.g. BANKS[3, 4], called *tree-like data-graph-based* KSORD (TD-KSORD), return a tuple connection tree. In this paper, we focus on TD-KSORD systems.

One of the most critical challenges in KSORD research is how to present the query results[1, 16]. This is not easy for the following reasons. Firstly, the results need to be semantically meaningful to users. However, a result which is a tuple or a tuple connection tree is not easy to be quickly understood by end users. Secondly, it is important to avoid overwhelming users with a huge number of

J.X. Yu, M. Kitsuregawa, and H.V. Leong (Eds.): WAIM 2006, LNCS 4016, pp. 385–396, 2006.

trivial results. However, lots of similar results are often produced, which makes users tired or confused. As we will see in section 5, previous works in KSORD do not solve these problems very well.

Organizing search results into clusters facilitates users' quick browsing through search results. Users can determine whether a group is relevant or not by examining simply its description: they can then explore just the relevant clusters and ignore the remaining ones, so that their browsing efficiency can be improved. This method has been widely used in presenting Web search results, while to the best of our knowledge, it has not been employed in KSORD research.

In this paper, we propose clustering to improve the presentation of search results, so as to improve the efficiency of users' browsing. Although many works about clustering have been done in related domains, traditional clustering methods are not applicable to KSORD results as is explained in section 5. In this paper, we focus on TD-KSORD systems, and propose a novel results clustering method named TreeCluster. It combines the structure and content information together and includes two steps of pattern clustering and keyword clustering. In the first step, we use labels to represent schema information of each result tree and cluster the trees into groups. The trees in each group are isomorphic. In the second step, we rank user keywords according to their frequencies in the database, and further partition the large groups based on the content of keyword nodes. Furthermore, we give each cluster a readable description, and present the description and each result tree graphically to help users understand the results more easily. Experimental results verify our methods' effectiveness and efficiency.

Organization: Section 2 introduces the basic concepts needed. Section 3 provides the detail of our solution and algorithms. The experimental results are shown in section 4. Section 5 reviews the related work. Finally, Section 6 concludes this paper.

2 Basic Concepts

We define some terms we will use in the following sections. They are based on [3] and have been adjusted slightly for simplification.

Definition 1 (Data Graph). Database can be represented as an undirected Data Graph G(V,E) which is composed of weighted nodes and weighted edges.

Nodes: For each tuple t in the database, the graph has a corresponding node $u_t \in V$. We will speak interchangeably of a tuple and the corresponding node in the graph.

Edges: For each pair of tuples t_1 and t_2 such that there is a foreign key from t_1 to t_2, the graph contains an undirected edge $< u_{t_1}, u_{t_2} >$.

Weights: Each node and edge is assigned a weight.

Definition 2 (Keyword Query). User's input is defined as a keyword query, which generally consists of n ($n \geq 1$) search terms $k_1, k_2, ..., k_n$.

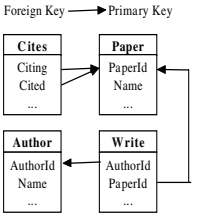

Fig. 1. DBLP Schema

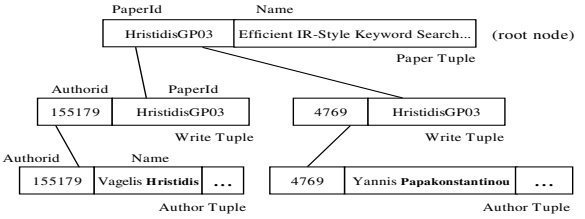

Fig. 2. An Example of result tree

A node is relevant to a search term if it contains the search term as part of an attribute value. It is called a **keyword node**. Generally, the first step of search algorithms is to locate the set of keyword nodes S_i that are relevant to k_i for each k_i in the query.

Definition 3 (Result Tree). An answer to a query is a rooted weighted tree containing at least one node from each S_i.

The relevance score of a result tree is computed from the weights of its nodes and edges. Result trees should be ranked in descending order of relevance score to meet the requirement of top-k query.

We call the root node of a result tree a **information node**, which connects all the keyword nodes, and strongly reflects the relationship among them.

For example, Figure 1 shows the schema of DBLP[14] dataset. Given a keyword query (*Hristidis, Papakonstantinou*), TD-KSORD systems find top-k result trees from the datagraph of DBLP. Figure 2 shows an example of one of the result trees, which is a subgraph of DBLP's data graph and means *Hristidis* and *Papakonstantinou* coauthor a paper.

3 TreeCluster

The problem we will solve is to find a clustering method to organize result trees into significant groups. First, we introduce the intuition of our method, and then describe the implementation, finally introduce the graphical user interface.

3.1 Intuition

We did much observation on different datasets, and found that many of the result trees were of the same pattern. For example, in DBLP, keyword query (*Jim Gray, Transaction*) may lead to many results. Some of them belong to the pattern that *Jim Gray* writes papers about *transaction*, while some of them belong to the one that *Jim Gray's* papers are cited by papers about *transaction*, and others may belong to the one that *Jim Gray's* papers cites papers about *transaction*. Thus, we can cluster all these results into various groups according to different patterns, and give a readable description for each group.

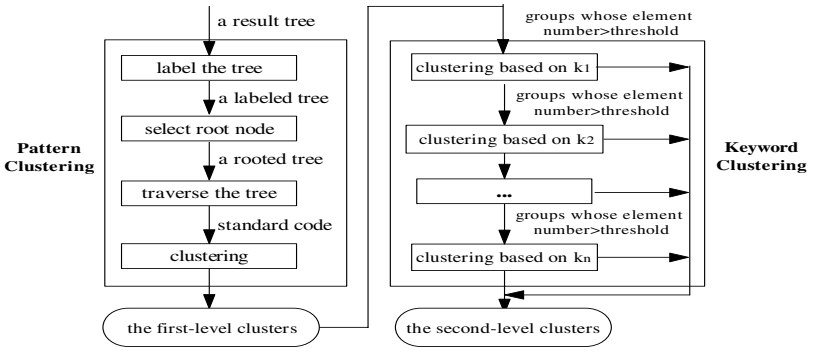

Fig. 3. Architecture of TreeCluster

Furthermore, we find that the resulting clusters in some patterns are quite large. So we decide to partition the large clusters further based on the content. From users' viewpoints, the most meaningful things are the keywords they input, so we can do partitioning based on the content of keyword nodes. The keywords should not be treated equally however. In fact, different keywords have different "frequencies" in the database. For instance, in DBLP, for keyword query (*Gray, Transaction*), *Gray* only appears in a few tuples, while *Transaction* appears in lots of tuples. We can partition large clusters based on the content of nodes relevant to low frequency keywords first. In this example, we partition a large group according to nodes relevant to *Gray* first. Thus result trees relevant to different "Gray"s, e.g. *Jim Gray* and *W.A.Gray*, are separated. Users need only examine the label (*Jim Gray*) or (*W.A.Gray*) for each subgroup to determine which one they are interested in instead of browsing through each result in the large group.

Figure 3 shows the framework of TreeCluster. It includes two steps, and produces two levels of clusters. After pattern clustering, we get the first-level groups, each of which corresponds to a kind of tree pattern. The large groups, whose numbers of elements exceed the threshold, will be processed by keyword clustering, after which, we get the second-level groups.

3.2 Pattern Clustering

Firstly, we cite definitions and conclusions about labeled trees from [9, 10], without detailed explanations due to space limitations.

Theorem 1. Two rooted ordered labeled trees are isomorphic if and only if their preorder traversal codes are equal.

Definition 4 (Standard Code). Let T be a rooted unordered labeled tree. All rooted ordered trees derived from T are named $T_1, T_2, ..., T_n$, whose preorder traversal codes are $S_1, S_2, ..., S_n$ respectively. We call the minimum code S_{min} of $S_1, S_2, ..., S_n$ the standard code of T.

Algorithm 1: GetStandardCode(t)

Global: special symbols '#' and '$' ('#' > '$' >all the label symbols)
Input: t: the root of a rooted unordered labeled tree T
Output: the standard code of T
St ← label(t)+"$"; // "+" means connecting
for *each edge e that comes from t to its sons* **do**
    St2 ← label(e); get another node n of e;
    insert St2+GetStandardCode(n)+"$" into set S;
end
sort strings in S in ascending order;
for *each string s in S* **do**
    append s to St;
end
return St+"#";

Theorem 2. Algorithm 1 computes the standard code of a rooted unordered tree correctly.

Theorem 3. Two rooted unordered labeled trees are isomorphic if and only if their standard codes are equal.

Now, we label the nodes and edges with schema information, so that we can express the pattern using traversal code of the tree. For an ordinary node (not keyword nodes), we may easily use the relation name it belongs to as its label. For a keyword node, things are more complex, because a keyword node may contain several keywords, and a keyword may appear in several attributes of a node. For an edge, what we concern is the primary-foreign key relationship. Thus we get the following rules.

Rule 1. Assume a node t, $t \in$ relation R. If t is an ordinary node, the label of t is [R]. If t is a keyword node, which contains keywords $k_1, k_2, ..., k_n$, and k_i is contained in attributes $A_{i_1}, A_{i_2}, ..., A_{i_{m_i}} (1 \leq i \leq n)$, then the label of t is $[Rk_1(A_{1_1}A_{1_2}...A_{1_{m_1}})...k_i(A_{i_1}A_{i_2}...A_{i_{m_i}})...k_n(A_{n_1}A_{n_2}...A_{n_{m_n}})]$.

Rule 2. Assume an edge $< t_1, t_2 >, t_1 \in$ relation $R_1, t_2 \in$ relation R_2, and assume the corresponding foreign key is $(A_1, ..., A_r)(A_i \in R_1, 1 \leq i \leq r)$, the corresponding primary key is $(B_1, ..., B_r)(B_i \in R_2, 1 \leq i \leq r)$, then the label of $< t_1, t_2 >$ is $\{(A_1, ..., A_r), (B_1, ..., B_r)\}$.

Because of the search mechanism in TD-KSORD systems, the roots of result trees in the same pattern may not be correspondent. For example, different result trees (the two authors coauthor different papers) in Figure 4 and Figure 2 have the same pattern, but their root nodes are not in correspondence. Therefore, we need to select a new root for each result tree to ensure the roots of trees in the same pattern are correspondent. In addition, such root nodes should contain as much information as possible. In this example, the root node of the tree in Figure 4 should be the "Paper Tuple".

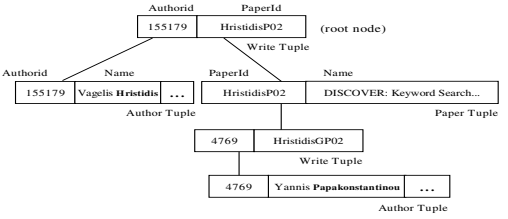

Fig. 4. A result tree in the same pattern with the tree in Figure 2

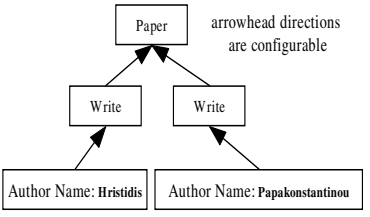

Fig. 5. An Example of the first-level cluster description

We consider firstly the nodes having the maximum degree, if there are many such nodes, we select those closest to the center of the tree. Usually, there is only one candidate node meeting the above two conditions. If there are more than one however, we use each of the candidate nodes in turn as the root and employ Algorithm 1 to compute the standard codes of the tree respectively. The one with the minimum standard code is selected as the information node. If there are more than one root node resulting in the minimum standard code, we can use any of them as the root node, because they produce the same standard code and do not affect the judging of isomorphism.

Now we get a rooted unordered labeled tree, we could use algorithm 1 to compute its standard code. According to Theorem 3, trees having the same standard codes are isomorphic and are clustered into a group. Thus we get the first-level clusters.

3.3 Keyword Clustering

We firstly rank the keywords according to their frequencies in the database, i.e. the number of keyword nodes which contain the specified keyword. Assume the new order is k_1, k_2,...,k_n. Then, we examine each group. If the number of elements in the group exceeds the threshold, we partition it firstly based on k_1, that is, if the contents of nodes in two trees relevant to k_1 are the same, the two trees are put into one group, otherwise separated into different groups. If the new groups still contain more than the threshold number of elements, we will continue to partition them based on k_2, and etc, until the number of elements in each cluster is less than the threshold or all the keywords are used up. Algorithm 2 shows the details.

3.4 GUI and Cluster Description

We build a graphic user interface for result representation, as demonstrated in Figure 6, in windows explorer style. For the results in each cluster, we rank them according to the relevant score in descending order. Furthermore, we get the maximum relevant score of each cluster, and rank clusters based on their maximum scores in descending order too.

In order to make the results semantically meaningful to users, we give a readable description for each cluster and present the description and each result graphically. Each tuple connection tree is presented in graph, as shown in Figure 2. The first-level cluster description mainly has the following characteristics. Firstly, it uses alias for database relations and attributes, so that database schema information is shielded to end users and improve the readability. Secondly, in order to focus on the pattern information, an ordinary node is only annotated with its relation alias, while a keyword node annotated with its relation alias, attribute alias and keyword itself. Thirdly, the direction of the edges between nodes can be configured in advance to provide more semantical meaning. Figure 5 is the cluster description of Figure 2 and Figure 4. Apparently, it can be understood quickly by users. For the second-level clusters, we label them with the keywords based on which the group is produced.

Algorithm 2: Group(S, k)

Global: THRESHOLD; KeyWord[ ]: an array of ranked keywords according to their frequencies in ascending order
Input: S: a group (set of trees) to be clustered; k: the index of current keyword
Output: set of the subgroups of S
if $k > KeywordNum$ **then** {insert S into V; return V;}
for *each tree t in S* **do**
    search set S2 in V, requiring the content of nodes relevant to KeyWord[k] of trees in S2 is the same as that of t;
    if *S2 exists* **then** {add t into S2;}
    else {NEW(S2), add t into S2, and insert S2 into V;}
end
for *each set S2 in V* **do**
    if $|S2| < THRESHOLD$ **then** { add S2 into V2;}
    else
        $V3 \leftarrow group(S2, k+1)$;
        for *each set S3 in V3* **do**
            add S3 into V2;
        end
    end
end
return V2;

4 Experiments

A search result clustering system is designed using Java, as shown in Figure 6. The system accepts query inputs from users and passes them to KSORD systems. Users can select one of the two result presentation manners: list or cluster. The former is the traditional method that simply presents ranked results in order, while the latter is this paper's work of presenting ranked results in clusters. As experiments demonstrates below, the latter is almost as fast as the former.

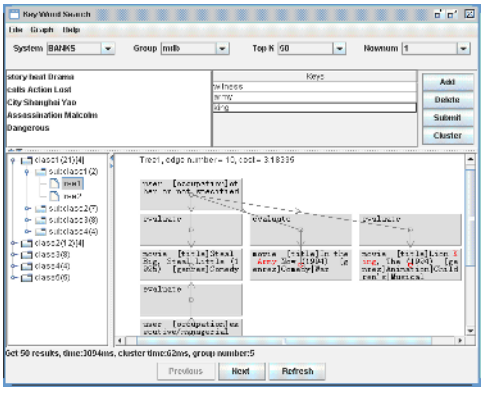

Fig. 6. The GUI of Search Result Clustering

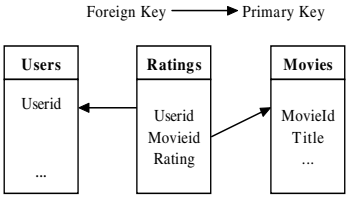

Fig. 7. MDB Schema

We conduct tests using Oracle9i on a AMD844*4 CPU and 4G memory Dawning server running Windows 2000 Advanced Server, using BANKS as KSORD system and it connects to Oracle9i through JDBC. In our figures, C-BANKS means BANKS using cluster as presentation manner, while L-BANKS means BANKS using list manner. For each test, we experiment on two real datasets, a subset of DBLP and a subset of MDB[15]. Our DBLP consists of about 497,000 nodes and 567,000 edges. Our MDB consists of about 506,000 nodes and 997,000 edges. The schema of MDB is shown in Figure 7.

For each experiment, we randomly generate 100 queries, and test the average effectiveness and efficiency. We partition the keywords extracted from the two datasets into three category according to their frequencies: high(H), medium(M), and low(L). We will show the experimental results of various patterns of keyword queries, although we only use keywords in medium and high frequency to do tests in order to meet the real-life case.

Due to space limitations, we always set the threshold of keyword clustering to 10 and do not report experimental results of other threshold. Apparently, as the threshold arise, the group number of the second-level will decrease.

4.1 Effectiveness

We call the average group number of the first-level **F-Num**, and use it to evaluate the effectiveness of pattern clustering. We use the number of overall groups including groups of the first-level that do not have subgroups and groups of the second-level to evaluate the overall effectiveness, and call it **O-Num**. Apparently, neither too many nor too few groups is good, and only medium F-Num and O-Num helps to improve users' browsing efficiency.

Number of Keywords. In Figure 8 and 9, we fix result number to 100 and vary the number of keywords from 2 to 6, to test F-Num and O-Num. We can see that in most cases, F-Num and O-Num are medium, which verifies our methods' effectiveness.

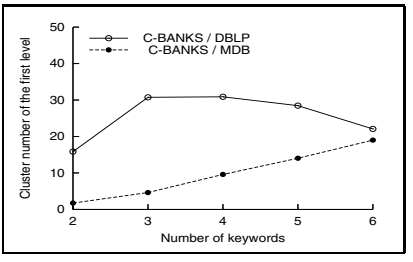

Fig. 8. Effectiveness(a): F_Num. Fix *Top-k* = 100, and vary KeywordNum.

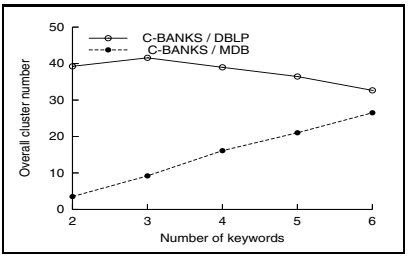

Fig. 9. Effectiveness(b): O_Num. Fix *Top-k* = 100, and vary KeywordNum.

In Figure 8, F-Num in DBLP is always larger than that in MDB, because the schema of MDB (primary-foreign key relationship) is simpler than that of DBLP, so that top-k results are likely in the same pattern. As keyword number increases, F-Num in MDB increases while that in DBLP decreases with over 3 keywords. The reason is that keyword frequencies in MDB are significantly lower than those in DBLP, so results that contain more keywords in MDB are not likely in the same pattern, while in DBLP, results containing more keywords are more likely produced by one Cartesian product[3] and thus in the same pattern.

It's easy to understand that if F-Num is small, the element number of each group is more likely to exceed the threshold and will be partitioned further by keyword clustering. Thus in Figure 9, although the varying trend of O-Num is similar to that of F-Num, it varies more gently.

Number of Results. In Figure 10 and 11, we fix keyword number to 3 and vary the number of returned results from 20 to 120. We can see F-Num and O-Num basically linearly increase as the top-k increases, which demonstrates our methods' good scalability.

Keyword Patterns. In Figure 12 and 13, we fix the number of keywords to 3, and the number of returned results to 100, and report 10 representative keyword patterns. For instance, pattern MML represents two medium frequency and one low frequency keywords.

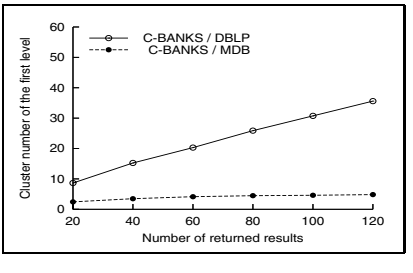

Fig. 10. Effectiveness(c): F_Num. Fix *KeywordNum* = 3, and vary Top-k.

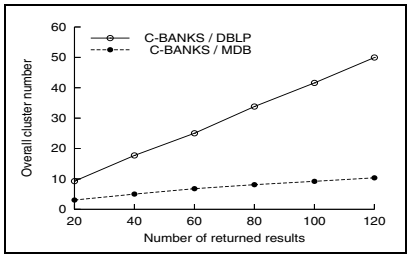

Fig. 11. Effectiveness(d): O_Num. Fix *KeywordNum* = 3, and vary Top-k.

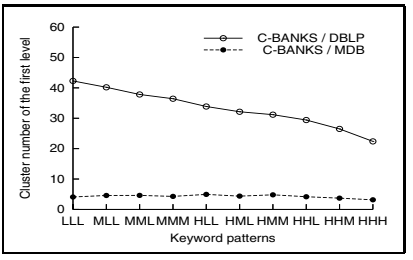

Fig. 12. Effectiveness(e): F_Num. Fix *KeywordNum* = 3, *Top-k* = 100, and vary Keyword Pattern.

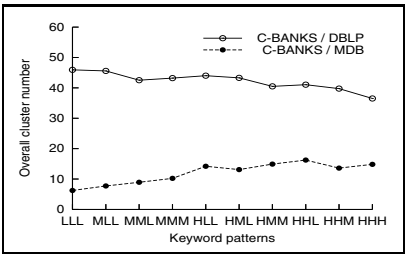

Fig. 13. Effectiveness(f): O_Num. Fix *KeywordNum* = 3, *Top-k* = 100, and vary Keyword Pattern.

In Figure 12, as keyword pattern contains higher frequency keywords, F-Num decreases in DBLP, which shows that keywords in higher frequency is more likely to produce results in the same pattern. F-Num varies a little in MDB because most of the keywords in MDB appear only a few times.

Figure 13 shows that higher frequency words play an import role in keyword clustering, because they appear in many tuples and the contents of these tuples are usually not equal.

Discussion. In general, simpler database schema and higher frequency keywords incline to decrease F-Num, while lower frequency keywords incline to increase F-Num. Higher frequency keywords incline to increase the group number of the second-level resulting in the increment of O-Num.

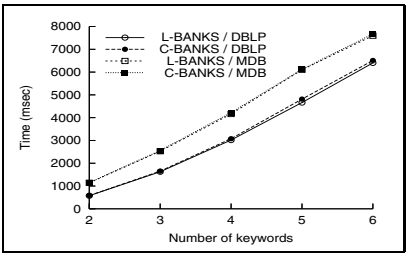

Fig. 14. Efficiency(a). Fix *Top-k* = 100 and vary KeywordNum.

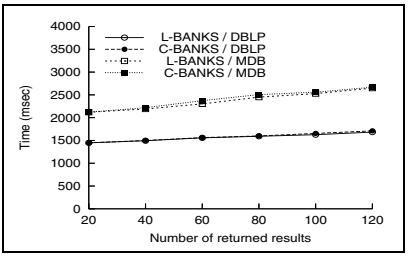

Fig. 15. Efficiency(b). Fix *KeywordNum* = 3 and vary Top-k.

4.2 Efficiency

We run the system using two result presentation manners (original list presentation and our cluster presentation) separately and compare their execution time. We can see from Figure 14 and 15 that two manners needs almost the same execution time. As keyword number varies from 2 to 6 or as the number of returned results varies from 20 to 120, clustering time increases only a little (the difference between C-BANKS/dataset and L-BANKS/dataset means the value of clustering time). The efficiency of BANKS on MDB is lower than that on

DBLP, which is also because keyword frequencies in MDB are lower than those in DBLP.

Usually, clustering search results always hurts the efficiency of systems, as previous works in Web search do. However, our method has slightly effect on original system efficiency.

5 Related Work

In KSORD research, many ways are used to present query results. BANKS[3] shows the query results in a nested table, based on which [13] improves the answer format by addressing readability. DbSurfer[12] uses tree-like structures to display all trails, while DataSpot[11] uses a distinguished answer node to represent a result. However, these works do not solve the problem of lots of similar results. In this paper, we organize the results into clusters and present them graphically to improve users' browsing efficiency.

[9] proposes a result classification method. In preprocessing, the system produces various patterns, and in processing a query, users select a particular pattern and the system searches the results matching the selected pattern. [13] mentions the similar idea, however with no implementation details. This method has to be implemented inside the search engine of a KSORD system, and closely bundled with the system. Our method can be implemented outside the system and applicable to various TD-KSORD systems.

Clustering results has been investigated in many works in the context of Web search. These works (e.g. [17, 18, 19]) are based on the content similarity and cluster documents into topically-coherent groups. Vivisimo[20] is a real demonstration of clustering Web search results. However, clustering methods used in Web are not applicable to KSORD. On the one hand, results of KSORD belong to a community (a professional database, such as DBLP or MDB), clustering based on content similarity usually can not get distinguished groups. On the other hand, information of RDBMS schema which is not available in Web search should be employed to instruct clustering.

[21] proposes to categorize the results of SQL queries, and generates multi-level category structure. However, according to the characteristics of our method, we only produce two levels of categorization, including the results of pattern clustering and keyword clustering respectively.

Traditional clustering research includes partitioning method, hierarchical method, density-based method, and etc[22]. Our method is different from them, aiming at the character of KSORD results. There are many works about judging isomorphism of rooted labeled trees. We directly cite the conclusions from [9, 10] without detailed explanations due to space limitations.

6 Conclusion and Future Work

In this paper, we proposed a novel clustering method named TreeCluster to organize search results of TD-KSORD system to improve users' browsing efficiency.

Furthermore, we generated readable cluster description, and presented the description and each result graphically to help users understand the results more easily. Experimental results verify effectiveness and efficiency of our method. This is the first proposal for clustering search results of KSORD.

In future work, we will detect more database schema information in the search process of KSORD and utilize it to improve the clustering results.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (No.60473069 and No.60496325).

References

1. Shan Wang and Kun-Long Zhang. Searching Databases with Keywords. Journal of Computer Science and Technology, Volume 20, No.1, January 2005.
2. A. Hulgeri, G. Bhalotia, C. Nakhe et al. Keyword Search in Databases. IEEE Data Engineering Bulletin, vol. 24, pages 22-32, 2001.
3. G. Bhalotia, A. Hulgeri, C. Nakhe et al. Keyword Searching and Browsing in Databases using BANKS. ICDE'02.
4. Varun Kacholia, Shashank Pandit, Soumen Chakrabarti et al. Bidirectional Expansion For Keyword Search on Graph Databases. VLDB'05, pages 505-516.
5. S. Agrawal et al. DBXplorer: A System For Keyword-Based Search Over Relational Databases. ICDE'02.
6. V. Hristidis et al. DISCOVER: Keyword Search in Relational Databases. VLDB'02.
7. V. Hristidis et al. Efficient IR-Style Keyword Search over Relational Databases. VLDB'03.
8. A. Balmin et al. ObjectRank: Authority-Based Keyword Search in Databases. VLDB'04.
9. Kun-Long Zhang. Research on New Preprocessing Technology for Keyword Search in Databases. PH.D thesis of Renmin University of China, 2005.
10. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.
11. S. Dar et al. DTL's DataSpot:Database Exploration Using Plain Language. VLDB'98.
12. R. Wheeldon et al. DbSurfer: A Search and Navigation Took for Relational Databases. The 21st Annual British National Conference on Databases, 2004.
13. B. Aditya et al. User Interaction in the BANKS System: A Demostration. ICDE'03, Demo.
14. DBLP Bibliography. http://www.informatik.uni-trier.de/ ley/db/index.html.
15. J. Riedl and J. Konstan. MoveLens. http://www.grouplens.org/.
16. V. Hristidis et al. Keyword Proximity Search on XML Graphs. ICDE'03.
17. Cutting D. R. et al. Constant Interaction-Time Scatter/Gather Browsing of Very Large Document Collections. SIGIR'93.
18. Zamir O. et al. Web Document Clustering: A Feasibility Demonstration. SIGIR'98.
19. Hua-Jun Zeng et al. Learning to Cluster Web Search Results. SIGIR'04.
20. Vivisimo clustering engine,(2004) http://vivisimo.com.
21. K.Chakrabarti et al. Automatic Categorization of Query Results. SIGMOD'04.
22. A.K. Jain et al. Data Clustering: A Review. ACM Computing Surveys, Vol 31, No.3, 1999: 264-323.