A Framework for Grouping and Summarizing Keyword Search Results

Orestis Gkorgkas[1], Kostas Stefanidis[2], and Kjetil Nørvåg[1]

[1] Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway
{orestis,Kjetil.Norvag}@idi.ntnu.no
[2] Institute of Computer Science, FORTH, Heraklion, Greece
kstef@ics.forth.gr

Abstract. With the rapid growth of the Web, keyword-based searches become extremely ambiguous. To guide users to identify the results of their interest, in this paper, we consider an alternative way for presenting the results of a keyword search. In particular, we propose a framework for organizing the results into groups that contain results with similar content and refer to similar temporal characteristics. Moreover, we provide summaries of results as hints for query refinement. A summary of a result set is expressed as a set of popular keywords in the result set. Finally, we report evaluation results of the effectiveness of our approach.

1 Introduction

Keyword-based search is extremely popular as a means for exploring information of interest without using complicated queries or being aware of the underlying structure of the data. Existing approaches for keyword search in relational databases use either the database schema (e.g., [1, 12]) or the given database instance (e.g., [5]) to retrieve tuples containing the keywords of a posed query. For example, consider the movie database instance depicted in Fig. 1. For the keyword query $Q = \{comedy, J. Davis\}$, the results are the *comedy* movies *Deconstructing Harry* and *Celebrity* both with *J. Davis*.

Given the huge volume of available data, keyword-based searches typically return overwhelming number of results. However, users would like to locate only the most relevant results to their information needs. Previous approaches mostly focus on ranking the results of keyword queries to help users retrieve a small piece of them. Such approaches include, among others, adapting IR-style document relevance ranking strategies (e.g., [11]) and exploiting the link structure of the database (e.g., [5]). Still, this flat ranked list of data items could not make it easy for the users to explore and discover important items relevant to their needs.

In this paper, we consider an alternative presentation of the results of the queries expressed through sets of keywords. In particular, we add some structure to the ranked lists of query results. Our goal is to minimize the browsing effort of the users when posing queries, help users receive a broader view of the query results and, possibly, learn about data items that they are not aware of.

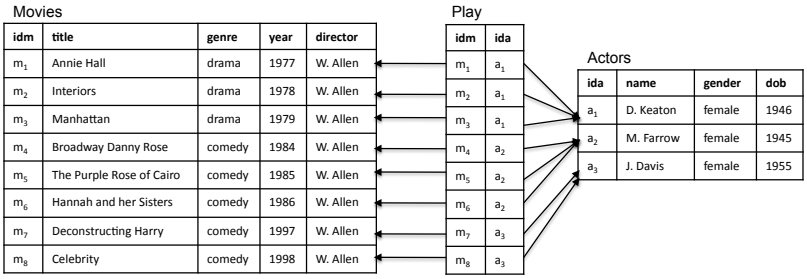

Fig. 1: Database instance.

Towards this direction, we organize the keyword query results into groups, trying to have groups that exhibit internal cohesion and external isolation. This way, it is easier for the users to scan the results of their queries. Our primary focus is on producing informative, expressive and meaningful groups containing results with similar content that refer to similar temporal characteristics. For example, assume the database instance of Fig. 1 and the keyword query $Q = \{W.\ Allen,\ female\}$. Intuitively, for this query, we can construct three groups of results; the first group refers to the movies *Annie Hall*, *Interiors* and *Manhattan*, the second group refers to the movies *Broadway Danny Rose*, *The Purple Rose of Cairo* and *Hannah and her Sisters* and the third one to the movies *Deconstructing Harry* and *Celebrity*. Each group contains movies with the same actress (*content similarity*) that are produced at the same time period (*temporal similarity*).

To help users refine their queries, we provide them with summaries over the groups of their queries results. The summary of a group presents the most important, in terms of popularity, keywords associated with the specific group of results. Abstractly speaking, for the above constructed groups, we may have the summaries $\{drama,\ D.\ Keaton\}$, $\{comedy,\ M.\ Farrow\}$ and $\{comedy,\ J.\ Davis\}$.

Finally, we evaluate the effectiveness of our approach. Our results indicate that users are more satisfied when grouping and summarizing of results are used.

In a nutshell, this paper makes the following contributions:

- It introduces a framework that offers a different way for presenting the results of keyword-based searches.
- It exploits the content of results along with their temporal characteristics to produce groups of results with similar content referring to the same time periods. Summaries for the groups of results are presented to users as hints for query refinement.
- It presents the results of a user study comparing our framework to a standard keyword search technique.

The rest of the paper is organized as follows. In Section 2, we introduce our framework for grouping and summarizing the results of keyword-based searches. In Section 3, we present our evaluation findings. Section 4 describes related work

and finally, Section 5 concludes the paper with a summary of our contributions and directions for future work.

2 Framework

Most approaches to keyword search (e.g., [1, 12]) exploit the dependencies in the database schema for answering keyword queries. Consider a database $\mathcal{D}$ with n relations $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$. We assume that some relations in $\mathcal{R}$ include, among other attributes, a time-related attribute B which represents the time that the entity described by the tuple was created. The *schema graph* $\mathcal{G}_D$ of a database $\mathcal{D}$ is a directed graph capturing the foreign key relationships in the schema. $\mathcal{G}_D$ has one node for each relation R_i and an edge $R_i \rightarrow R_j$, if and only if, R_i has a set of foreign key attributes referring to the primary key attributes of R_j. We refer to the undirected version of the schema graph as $\mathcal{G}_U$.

Let W be the potentially infinite set of all keywords. A keyword query Q consists of a set of keywords, i.e., $Q \subseteq W$. Typically, the result of a keyword query is defined with regards to *joining trees of tuples* (JTTs), which are trees of tuples connected through primary to foreign key dependencies [1, 5, 12].

Our goal in this paper is twofold; first, we focus on organizing into groups the results of a keyword query based on their content similarity and the similarity on the values of their time-related attributes and then, we highlight the important keywords in the produced groups of results.

We start this section with a short introduction to keyword search and then present our approach for organizing the keyword query results in time-dependent groups. Finally, we describe our method for offering the important keywords in the constructed groups.

2.1 Keyword Search

This section gives some preliminaries on keyword search, starting by defining the brick of a keyword query result, i.e., the joining tree of tuples.

Definition 1 (Joining Tree of Tuples (JTT)). *Given an undirected schema graph $\mathcal{G}_U$, a joining tree of tuples (JTT) is a tree of tuples T, such that, for each pair of adjacent tuples t_i, t_j in T, $t_i \in R_i$, $t_j \in R_j$, there is an edge (R_i, R_j) $\in \mathcal{G}_U$ and it holds that ($t_i \bowtie t_j$) $\in$ ($R_i \bowtie R_j$).*

For example, (m_7, *Deconstructing Harry, comedy, 1997, W. Allen*) – (m_7, a_3) – (a_3, *J. Davis, female, 1955*) represents a JTT for the keyword query Q = {*comedy, J. Davis*}. The size of a JTT is equal to the number of its tuples. In this case the aforementioned JTT has a size equal to 3.

Total JTT: A JTT T is *total* for a keyword query Q, if and only if, every keyword of Q is contained in at least one tuple of T.

Minimal JTT: A JTT T that is total for a keyword query Q is also *minimal* for Q, if and only if, we cannot remove a tuple from T and get a total JTT for Q.

We can now define the result of a keyword query as follows:

Definition 2 (Query Result). *Given a keyword query Q, the result $Res(Q)$ of Q is the set of all JTTs that are both total and minimal for Q.*

We use our movies example (Fig. 1) to briefly describe basic ideas of existing keyword query processing. For instance, the query $Q = \{comedy, J. Davis\}$ with result $Q = \{comedy, J. Davis\}$ consists of the JTTs: (i) (m_7, *Deconstructing Harry, comedy, 1997, W. Allen*) – (m_7, a_3) – (a_3, *J. Davis, female, 1955*) and (ii) (m_8, *Celebrity, comedy, 1998, W. Allen*) – (m_8, a_3) – (a_3, *J. Davis, female, 1955*). Each JTT in the result corresponds to a tree at schema level. That is, both of the above trees correspond to the schema level tree $Movies^{\{comedy\}} - Play^{\{\}} - Actors^{\{J.Davis\}}$, where each R_i^X consists of the tuples of R_i that contain all keywords of X and no other keyword of Q. Such sets are called *tuple sets* and the schema level trees are called *joining trees of tuple sets* (JTSs).

Several algorithms in the research literature aim at constructing such trees of tuple sets for a query Q as an intermediate step of the computation of the final results (e.g. [1, 12]). We adopt the approach of [12] in which all JTSs with size up to l are constructed. In particular, given a query Q, all possible tuple sets R_i^X are computed, where $R_i^X = \{t \mid t \in R_i \wedge \forall a_x \in X$, t contains $a_x \wedge \forall a_y \in Q \backslash X$, t does not contain $a_y\}$. After selecting a random query keyword a_z, all tuple sets R_i^X for which $a_z \in X$ are located. These are the initial JTSs with only one node. Then, these trees are expanded either by adding a tuple set that contains at least another query keyword or a tuple set for which $X = \{\}$ (free tuple set). These trees can be further expanded. JTSs that contain all query keywords are returned, while JTSs of the form $R_i^X - R_j^{\{\}} - R_i^Y$, where an edge $R_j \rightarrow R_i$ exists in the schema graph, are pruned, since JTTs produced by them have more than one occurrence of the same tuple for every instance of the database.

2.2 Keyword Search Result Vector Representation

Our effort focuses on grouping results based on their content and some temporal information associated with them. Regarding the content of a JTT, we may think of a JTT as the equivalent of a "document". Then, the textual content of a JTT can be represented by a term-vector. For a query Q with result $Res(Q)$, let $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. The importance score $x_{i,j}$ of a keyword a_i in $\mathcal{A}$ for the JTT T_j of $Res(Q)$, is defined with respect to the TF-IDF model [7]. Specifically, for each a_i in $\mathcal{A}$ for T_j, $x_{i,j}$ is equal to: $x_{i,j} = tf_{i,j} * log(N/df_i)$, where $tf_{i,j}$ is the number of occurrences of a_i in the JTT T_j and df_i is the number of tuples in $\mathcal{D}$ that contain a_i. N is the maximum df in the database. Then, a JTT-vector for a specific JTT is:

Definition 3 (JTT-vector). *Let Q be a keyword query with query result $Res(Q)$ and $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. The JTT-vector of a JTT T_j in $Res(Q)$ is a vector $u_{T_j} = \{(a_1, x_{1,j}), \ldots, (a_m, x_{m,j})\}$, where $a_i \in \mathcal{A}$, $|\mathcal{A}| = m$, and $x_{i,j}$ is the importance score of a_i for T_j, $1 \leq i \leq m$.*

Many times, two JTTs may contain very similar information. Next, we will exploit similarities between JTTs in order to construct groups of similar results.

2.3 Finding Groups of Keyword Search Results

In this work, we consider that each database relation includes in its schema a time-related attribute B[3]. Then, for a tuple t_i of a relation R_j, $1 \leq j \leq n$, we refer to the value $t_i[R_j.B]$ as the *age* of u. Naturally time-related attributes of the database relations may vary. For instance, for a relation with *movies* consider the *production year* as a time-related attribute or for a relation with *actors* the *date of birth*. For two tuples t_i, t_x of the relations R_j, R_y, we say that t_i is more recent than t_x, if and only if, $t_i[R_j.B] > t_x[R_y.B]$, $1 \leq j, y \leq n$.

Given a joining tree of tuples T, we define its *age* with respect to the age of the tuples appearing in the tree. In particular, the age of T is determined by the age of the most recent of its tuples. The motivation behind this, is that before the existence of the entity described in the most recent tuple the tree did not exist. For example, let $Movies.B$ be the attribute *year* of the relation $Movies$ and $Actors.B$ be the attribute *dob* of the relation $Actors$. Furthermore, consider that each tuple t_i of the relation $Play$ has value $t_i[Play.B] = 0$. Then, the age of the JTT (m_7, *Deconstructing Harry, comedy, 1997, W. Allen*) – (m_7, a_3) – (a_3, *J. Davis, female, 1955*) is 1997 which is the production year of the movie. Formally:

Definition 4 (Age of JTT). *Given a JTT T with tuples $t_1 \in R_{j_1}$, ..., $t_p \in R_{j_p}$, $1 \leq j_1, j_p \leq n$, the age of T, age_T, is:*

$$age_T = \max_{1 \leq i \leq p} \{t_i[R_{j_i}.B]\}$$

Our goal here is to detect groups of JTTs. Each group contains JTTs that: (i) have similar content and (ii) are continuous in time, which means that their *age* values increase. A straightforward way for quantifying the similarity between two JTTs, is to use a cosine-based definition of similarity, which measures the similarity between their corresponding vectors.

Definition 5 (Cosine JTT Similarity). *Given two JTTs T_1 and T_2 with vectors $u_{T_1} = \{(a_1, x_{1,1}), \ldots, (a_m, x_{m,1})\}$ and $u_{T_2} = \{(a_1, x_{1,2}), \ldots, (a_m, x_{m,2})\}$, respectively, the cosine JTT similarity between T_1 and T_2 is:*

$$sim_c(T_1, T_2) = \frac{u_{T_1} \cdot u_{T_2}}{||u_{T_1}||||u_{T_2}||} = \frac{\sum_{i=1}^{m} x_{i,1} \times x_{i,2}}{\sqrt{\sum_{i=1}^{m} (x_{i,1})^2} \times \sqrt{\sum_{i=1}^{m} (x_{i,2})^2}}$$

Given the similarity between JTTs, we focus on the grouping process. A group of JTTs is expressed as a set of JTTs. The JTTs of a group G_j define a time interval described by two time instances $G_j.s$ and $G_j.e$; $G_j.s$ denotes the starting point of the interval and corresponds to the age of the oldest JTT in the group, while $G_j.e$ denotes the ending point of the interval and corresponds to the age of the most recent JTT. For example, for a group G_j consisting of the

[3] If a relation does not contain time-related data we consider $B = 0$ for all tuples in the relation.

JTTs (i) (m_1, *Annie Hall, drama, 1977, W. Allen*) – (m_1, a_1) – (a_1, *D. Keaton, female, 1946*), (ii) (m_2, *Interiors, drama, 1978, W. Allen*) – (m_2, a_1) – (a_1, *D. Keaton, female, 1946*) and (iii) (m_3, *Manhattan, drama, 1979, W. Allen*) – (m_3, a_1) – (a_1, *D. Keaton, female, 1946*), $G_j.s = 1977$ and $G_j.e = 1979$.

Similarly to the JTT-vector, we define the Group-vector which describes the content of a group. In particular, the Group-vector of a group G_j is an aggregation of all vectors of the JTTs belonging to G_j. For a query Q with result $Res(Q)$, let $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$ and G_j be a group of JTTs in $Res(Q)$. The importance score $s_{i,j}$ of a keyword a_i in $\mathcal{A}$ for the group G_j is equal to: $s_{i,j} = Aggr_{T_w \in G_j}(x_{i,w})$, where $Aggr$ defines the *average*, *sum*, *maximum* or *minimum* of the values $x_{i,w}$ of the JTTs of G_j.

Definition 6 (Group-vector). *Let G_j be a group of JTTs belonging to the query result $Res(Q)$ of a query Q and $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. The Group-vector of G_j is a vector $u_{G_j} = \{(a_1, s_{1,j}), \ldots, (a_m, s_{m,j})\}$, where $a_i \in \mathcal{A}$, $|\mathcal{A}| = m$, and $s_{i,j}$ is the importance score of a_i for G_j, $1 \leq i \leq m$.*

Given the query result $Res(Q)$ of a query Q, our aim is to partition the JTTs of $Res(Q)$ into non-overlapping groups. Our definition for non-overlapping groups takes into account both time and content overlaps. Specifically, two groups are: (i) non-overlapping with respect to time, if their time intervals are disjoint and (ii) non-overlapping with respect to content, if they do not contain common JTTs.

Definition 7 (Non-overlapping Groups). *Let G_i, G_j be two groups of JTTs with time-intervals $[G_i.s, G_i.e]$, $[G_j.s, G_j.e]$. G_i, G_j are non-overlapping groups, if and only if: (i) ($G_i.s > G_j.e$ and $G_i.s > G_j.s$) or ($G_j.s > G_i.e$ and $G_j.s > G_i.s$), and (ii) $G_i \cap G_j = \emptyset$.*

To partition the joining trees of tuples into non-overlapping groups, we employ a bottom-up hierarchical agglomerative clustering method. Initially, the *JTT Partitioning Algorithm* (Algorithm 1) places each JTT in a cluster of its own. Then, at each iteration, it merges the two most similar clusters. The similarity between two clusters is defined as the minimum similarity between any two JTTs that belong to these clusters (*max linkage*). That is, for two clusters, or groups, G_1, G_2: $sim(G_1, G_2) = \min_{T_i \in G_1, T_j \in G_2}\{sim_c(T_i, T_j)\}$.

Clearly, two clusters, or groups, G_1, G_2 can be merged if they are non-overlapping groups. But this is not enough. For constructing groups with JTTs with growing age values there is also a need to ensure that, for the groups G_1, G_2, there is no other group G_3 with time interval between the time intervals of G_1 and G_2. We refer to such groups as merge-able groups. Formally:

Definition 8 (Merge-able Groups). *Let G_i, G_j be two groups of JTTs with time-intervals $[G_i.s, G_i.e]$, $[G_j.s, G_j.e]$. G_i, G_j are merge-able groups, if and only if: (i) G_i, G_j are non-overlapping groups, and (ii) $\nexists G_p$ with time interval $[G_p.s, G_p.e]$, such that, the groups G_i, G_p and G_p, G_j are non-overlapping, and ($G_p.s > G_i.e$ and $G_j.s > G_p.e$) or ($G_p.s > G_j.e$ and $G_i.s > G_p.e$)*

Algorithm 1 JTT Partitioning Algorithm

Input: A set of JTTs.
Output: A set of groups of JTTs.

```
1: Create a group for each JTT;
2: Repeat
3:   i = 1;
4:   Locate the two merge-able groups with the maximum similarity;
5:     If there are no merge-able groups or only one group existsthen
6:       End loop;
7:     Else
8:       Merge the two groups;
9:       Compute K_i, C_i;
10:      i++;
11: Select the partitioning that constructs K* groups;
```

Thus, in overall, we proceed in merging two groups only if the groups are merge-able. The algorithm stops either when a single cluster containing all the JTTs of $Res(Q)$ has already produced or when no more clusters can be merged. As a final step, the algorithm selects to return the clusters of the iteration that present the maximum clustering quality. The clustering quality C_i, computed after merging the two clusters of a specific iteration i, is:

$$C_i = \sum_{j=1}^{K_i} \sum_{\forall T_p \in G_j} u_{T_p} \cdot u_{G_j} \tag{1}$$

where K_i is the number of clusters after the merging operation of iteration i. The selected iteration is the one that constructs K^* clusters, such that:

$$K^* = argmax_i(C_i - \lambda K_i) \tag{2}$$

where λ is a penalty for each additional cluster.

Algorithm 1 presents a high level description of the *JTT Partitioning Algorithm*. Although it is possible to pre-specify the number of clusters K^* and directly select to return the clusters of the iteration that produces the K^* ones, we opt for following the above described procedure to ensure high clustering quality, even if the resulting processing cost is high.

We illustrate our approach with the following example. Assume the keyword query $Q = \{$*W. Allen, female*$\}$. For the database instance of Fig. 1, the result $Res(Q)$ consists of the JTTs:
(i) T_1: (m_1, *Annie Hall, drama, 1977, W. Allen*) - (m_1, a_1) - (a_1, *D. Keaton, female, 1946*), (ii) T_2: (m_2, *Interiors, drama, 1978, W. Allen*) - (m_2, a_1) - (a_1, *D. Keaton, female, 1946*), (iii) T_3: (m_3, *Manhattan, drama, 1979, W. Allen*) - (m_3, a_1) - (a_1, *D. Keaton, female, 1946*), (iv) T_4: (m_4, *Broadway Danny Rose, comedy, 1984, W. Allen*) - (m_4, a_2) - (a_2, *M. Farrow, female, 1945*), (v) T_5: (m_5, *The Purple Rose of Cairo, comedy, 1985, W. Allen*) - (m_5, a_2) - (a_2, *M. Farrow, female, 1945*), (vi) T_6: (m_6, *Hannah and her Sisters, comedy, 1986, W. Allen*) - (m_6, a_2) - (a_2, *M. Farrow, female,*

1945), (vii) T_7: (m_7, *Deconstructing Harry, comedy, 1997, W. Allen*) - (m_7, a_3) - (a_3, *J. Davis, female, 1955*) and (viii) T_8: (m_8, *Celebrity, comedy, 1998, W. Allen*) - (m_8, a_3) - (a_3, *J. Davis, female, 1955*),
with ages 1977, 1978, 1979, 1984, 1985, 1986, 1997 and 1998, respectively. Applying the *JTT Partitioning Algorithm* results in producing three groups G_1, G_2 and G_3 with trees $\{T_1, T_2, T_3\}$, $\{T_4, T_5, T_6\}$ and $\{T_7, T_8\}$.

2.4 Summaries of Keyword Query Results

In this section, we describe the notion of group summaries that put in a nutshell the results within groups of keyword searches. In general, group summaries provide hints for query refinement and can lead to discoveries of interesting results that a user may be unaware of.

Let $Res(Q)$ be the query results of a query Q and $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. Let also $G_1, \ldots, G_z$ be the groups of JTTs produced for $Res(Q)$. Our goal is to compute an importance score $s_{i,j}$ for each keyword a_i in $\mathcal{A}$ for each group G_j, $1 \leq j \leq z$. Then, for each group G_j, the top-k keywords, that is, the k keywords with the highest importance scores are used as a summary of the JTTs in G_j. Formally:

Definition 9 (Group Summary). *Let G_j be a group of JTTs belonging to the query result $Res(Q)$ of a query Q and $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. The group summary S_{G_j}, $S_{G_j} \subseteq \mathcal{A}$, of G_j is a set of k keywords, such that, $s_{i,j} \geq s_{p,j}$, $\forall a_i \in S_{G_j}$, $a_p \in \mathcal{A} \backslash S_{G_j}$.*

For example, for the keyword query $Q = \{$*W. Allen, female*$\}$, the group summaries of the produced groups G_1, G_2 and G_3, for $k = 2$, are $S_{G_1} = \{$*drama, D. Keaton*$\}$, $S_{G_2} = \{$*comedy, M. Farrow*$\}$ and $S_{G_3} = \{$*comedy, J. Davis*$\}$.

To provide users with more detailed summaries that include some information about the schema of the results, we extend the notion of group summaries to take into account the relations that a keyword belongs to. Specifically, for each group G_j, instead of reporting the set of the k keywords with the highest importance scores, we report these keywords along with their associated relations. This way, users obtain an overview about the possible origination and meaning of the keywords. We refer to these summaries as enhanced group summaries. Formally:

Definition 10 (Enhanced Group Summary). *Let G_j be a group of JTTs and S_{G_j} be the corresponding group summary of G_j with keywords $a_1, \ldots, a_k$. The enhanced group summary $\mathcal{E}_{G_j}$ of G_j is a set of k pairs of the form (a_i, P_i), such that, there is one pair $\forall a_i \in S_{G_j}$ and P_i is the set of relations that contain a_i for the JTTs of G_j.*

Returning to our previous example, the enhanced group summaries of G_1, G_2 and G_3 are represented as $\mathcal{E}_{G_1} = \{($*drama,* $\{$*Movies*$\})$, ($D.$ *Keaton,* $\{$*Actors*$\})\}$, $\mathcal{E}_{G_2} = \{($*comedy,* $\{$*Movies*$\})$, ($M.$ *Farrow,* $\{$*Actors*$\})\}$ and $\mathcal{E}_{G_3} = \{($*comedy,* $\{$*Movies*$\})$, ($J.$ *Davis,* $\{$*Actors*$\})\}$, respectively.

Based on the summaries of the produced groups of results, we define the summary of the query result as a whole, as follows:

Definition 11 (Query Result Summary). *Let $G_1, \ldots, G_z$ be the groups of JTTs produced for the query result $Res(Q)$ of a query Q. The query result summary $\mathcal{S}_Q$ is a set of z group summaries, $\mathcal{S}_Q = \{\mathcal{S}_{G_1}, \ldots, \mathcal{S}_{G_z}\}$, such that, $\mathcal{S}_{G_j}$ is either the group summary S_{G_j} or the enhanced group summary $\mathcal{E}_{G_j}$ of G_j, $1 \leq j \leq z$.*

That is, for $Q = \{$*W. Allen, female*$\}$, the query result summary $\mathcal{S}_Q$ taking into account the group summaries is $\{\{$*drama, D. Keaton*$\}$, $\{$*comedy, M. Farrow*$\}$, $\{$*comedy, J. Davis*$\}\}$, while for the enhanced group summaries we have the summary $\{($*drama,* $\{$*Movies*$\})$, $($*D. Keaton,* $\{$*Actors*$\})$, $($*comedy,* $\{$*Movies*$\})$, $($*M. Farrow,* $\{$*Actors*$\})$, $($*comedy,* $\{$*Movies*$\})$, $($*J. Davis,* $\{$*Actors*$\})\}$.

We could also consider other versions for summaries. For instance, assume that the importance of each keyword is computed separately for each relation. Then, we may report important keywords with respect to their relation-specific scores or keywords for relations of high user interest.

Summary-based Exploratory Keyword Queries. Besides presenting summaries to the users and offering, this way, a side mean for further exploration, we also plan to use the summaries to directly discover interesting pieces of data that are potentially related to the users' information needs. Specifically, to locate such related information, special-purpose queries, called *summary-based exploratory keyword queries*, can be constructed. The focus of these queries is on retrieving results highly correlated with the results of the original users queries.

Our plan is to employ the keywords of summaries to emerge new interesting results. An exploratory keyword query for a query Q will consist of a set of keywords, that is a subset of the keywords in a group summary G_j of Q, that frequently appear together in the JTTs of G_j. There are also other ways for constructing exploratory queries that qualify different properties. For example, sets of keywords that frequently appear in the result and, at the same time, rarely appear in the database ensure high surprise, or unexpectedness, as a measure of interestingness, as surprise used in the data mining literature (e.g., [17]). Recently, exploratory queries are used for exploration in relational databases through recommendations [8].

3 Evaluation

To demonstrate the effectiveness of grouping and summarizing keyword search results, we conducted an empirical evaluation of our approach using a real movie dataset[4] with 30 volunteers with a moderate interest in movies. The schema of our database is shown in Fig. 2 while the size of the database is 1.1 GB.

We run our experiments for queries of different sizes, i.e., number of keywords, and keywords of a different selectivity. We presented the results to the participants using two methods, 1) without any grouping (baseline method) and 2) with groups produced by our approach (grouped method). In the baseline method, for each query, we presented an enhanced group summary of the whole

[4] http://www.imdb.com/interfaces

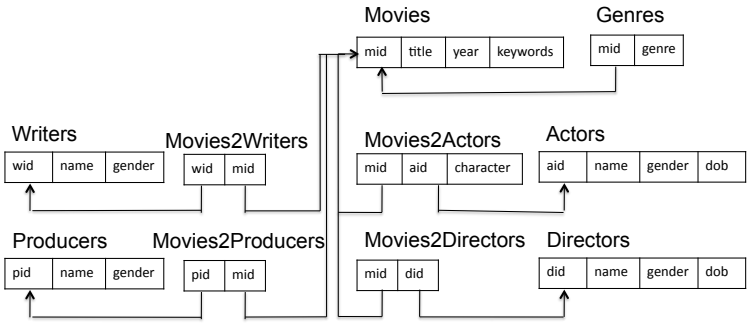

Fig. 2: Movies database schema.

result-set, considering the whole result-set as one group. To help the users understand the context of the significant terms we presented them also the attribute value in which a significant term appeared in. We give also the participants the ability to examine the set of produced JTTs. The results, i.e., the JTTs, are ranked based on their size that corresponds to the relevance of the trees to the query. In the grouped method, the participants are initially presented the groups of JTTs which were formed on the same results that were presented in the baseline. The groups are indicated to the participants by the time period each group covers. When a participant focuses on period he/she is provided with the summary of the group's content and the results belonging to that group.

The participants were asked to evaluate the quality of the results. For characterizing the quality, we use four measures: (i) *group coherence*, which evaluates the similarity of the results content inside a group , (ii) *baseline summary quality*, which evaluates how descriptive is the summary of the baseline method for the whole result set, (iii) *group summary quality*, which evaluates how descriptive is the summary of each group, and (iv) *usefulness evaluation*, that evaluates if the participant found the grouping method more helpful than the baseline method.

For grading the grouping, the participants were asked to evaluate for each group if the movies in the group fit well together. We used 3 values: not coherent (0), quite coherent (1), and very coherent (2). The users were also asked for each summary if it was descriptive of the result and if it was helpful for them to understand the content of the results. The summaries were also graded using three values: not descriptive (0), quite descriptive (1), and very descriptive (2). The degree of overall usefulness was graded with two values: our method is not helpful (0), and our method is helpful (1). Each query was evaluated by at least 3 participants while 95% of the queries were evaluated by at least 4 and 75% by at least 8. On average there were 8 evaluators per query.

Group coherence. Table 1 shows the average values of the coherence measure for each query as they were estimated by the participants. According to the average group coherence value, that is 1.52, the participants found the grouping of the results to be meaningful and helpful for them to understand the results.

Table 1: Group coherence evaluation for each query.

Query	Average group coherence
"Daniel Craig" movies	1.50
"James Bond" movies	1.50
"James Bond" male actors	1.50
"Woody Allen" female actors	1.27
"Clint Eastwood" movies	1.50
"Peter Jackson" male actors	1.75
"Peter Jackson" movies	1.33
"Denzel Washington" Action	1.88
"Julia Roberts" Comedy	1.71
"Julia Roberts" movies	1.75
"Kevin Spacey" drama	1.27
"Jack Nicholson" female actors	1.44
"Al Pacino" movies	1.45
"Al Pacino" male actors	1.50
"Al Pacino" directors	1.50
"Stanley Kubrick" actors	1.75
"Stanley Kubrick" movies	1.60
"Lord of The Rings" Tolkien	1.30
"Robert De Niro" directors	1.60
"Francis Ford Coppola" male actors	1.75

Summary quality. Table 2 reports the average values of the quality measures for each query (we omit the detailed per person scores due to space limitations). As it can be seen, in 90% of the queries the quality of group summaries was better than (or equal to) the quality of the baseline summary according to the participants. This comes to complete accordance with the percent of participants (85%) who found our approach helpful. We can also draw the conclusion that while in all queries the majority of participants found the grouped summaries to be quite or very descriptive, the baseline summary was evaluated as quite or very descriptive in only 30% of the queries.

Time overhead. Finally, we study the overall impact of grouping and summarizing keyword search results in terms of time overhead for the above query examples. In particular, we measured the time needed to build the JTTs and the time needed for creating the groups and summaries. The additional computational cost of our approach is small in comparison with the generation of the actual keyword search results. On average, the additional time consumed for creating the summaries and groups was a magnitude smaller than the JTT building time, and in no case was the creation of groups and summaries significantly more expensive in terms of time than the building of the JTTs. For example, the time overhead for the query {Stanley Kubrick, movies} is 3.7% and for the query {Francis Ford Coppola, male actors} is 9.5%.

Query	Baseline summary quality	Group summary quality	Usefulness
"Daniel Craig" movies	0.75	1.50	0.75
"James Bond" movies	0.70	1.60	0.90
"James Bond" male actors	1.00	1.50	0.75
"Woody Allen" female actors	0.82	1.45	0.82
"Clint Eastwood" movies	1.36	1.57	0.86
"Peter Jackson" male actors	0.50	1.75	1.00
"Peter Jackson" movies	1.67	1.67	0.67
"Denzel Washington" Action	1.13	1.88	1.00
"Julia Roberts" Comedy	0.88	1.43	0.86
"Julia Roberts" movies	1.13	1.75	0.88
"Kevin Spacey" drama	0.64	1.28	0.82
"Jack Nicholson" female actors	0.22	1.56	0.89
"Al Pacino" movies	1.00	1.45	0.82
"Al Pacino" male actors	0.50	1.63	0.88
"Al Pacino" directors	0.50	1.50	1.00
"Stanley Kubrick" actors	1.00	2.00	1.00
"Stanley Kubrick" movies	1.50	1.30	0.80
"Lord of The Rings" Tolkien	1.36	1.20	0.60
"Robert De Niro" directors	0.30	1.60	0.90
"Francis Ford Coppola" male actors	1.00	2.00	1.00

Table 2: Summary quality evaluation for each query.

4 Related Work

Keyword search in relational databases has been the focus of much current research. Schema-based approaches (e.g., [1, 12]) use the schema graph to generate join expressions and evaluate them to produce tuple trees. Instance-based approaches (e.g., [5]) represent the database as a graph in which there is a node for each tuple. Results are provided directly by using a Steiner tree algorithm. Based on [5], several more complex approaches have been proposed (e.g., [10, 13]). There have also been proposals for providing ranked keyword retrieval, which include incorporating IR-style relevance ranking [11], authority-based ranking [3], automated ranking based on workload and data statistics of query answers [6] and preference-based ranking [18, 20].

Our approach is different, in that, we propose grouping keyword search results to help users receive the general picture of the results of their queries. A comparison between a flat ranked list of results and a clustering web search interface shows that the users of the clustering approach view more documents and spend less time per document [21]. However, the relevance of the viewed documents is unknown. [15] presents an approach for clustering keyword search results based on common structure patterns without taking into account the aspect of time. Recently, [2] introduces a prototype and framework for interactive clustering of query results. This technique is applied in document collections, while our work focuses on structured data.

Summaries of keyword queries results resemble the notion of *tag clouds*. A tag cloud is a visual representation for text data. Tags are usually single words, alphabetically listed and in different font size and color to show their importance[5]. Tag clouds have appeared on several Web sites, such as Flickr and del.icio.us. With regard to our approach for summaries, *data clouds* [14] are the most relevant. This work proposes algorithms that try to discover good, not necessarily popular, keywords within the query results. Our approach follows a pure IR technique to locate important, in terms of popularity, keywords. From a database perspective, [9] introduces the notion of *object summary* for summarizing the data in a relational database about a particular *data subject*, or keyword. An object summary is a tree with a tuple containing the keyword as the root node and its neighboring tuples containing additional information as child nodes.

Finally, our work presents some similarities with faceted search (e.g., [4, 16]). Faceted search is an exploration technique that provides a form of navigational search. In particular, users are presented with query results classified into multiple categories and can refine the results by selecting different conditions. Our approach is different in that we do not tackle refinement. [8, 19] present a different way for database exploration by recommending to users items that are not part of the results of their query but appear to be highly related to them. Such items are computed based on the most interesting sets of attribute values that appear in the results of the original query. The interestingness of a set is defined based on its frequency in the results and the database.

5 Conclusions

In this paper we propose a framework for organizing the keyword search results into groups that contain results with similar content that refer to similar temporal characteristics. We employ summaries of results to help users refine their searches. A summary of a result set is expressed as a set of important attribute values in the result set. In addition we evaluate the effectiveness of our approach. Our usability results indicate that users are more satisfied when results are organized with respect to content and time than when results are simply ordered with respect to relevance.

Clearly, there are many directions for future research. Our plans include studying different functions for computing similarities between JTTs, as well as more efficient clustering algorithms. Finally, in this paper, we compute summaries based on popularity; other properties, such as the dependence on the query, should also be sought for.

Acknowledgments

The work of the second author is supported by the project "IdeaGarden" funded by the Seventh Framework Programme under grand n^o 318552.

[5] http://en.wikipedia.org/wiki/Tag_cloud

References

1. S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *Proc. of ICDE*, pages 5–16, 2002.
2. D. C. Anastasiu, B. J. Gao, and D. Buttler. A framework for personalized and collaborative clustering of search results. In *Proc. of CIKM*, pages 573–582, 2011.
3. A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *Proc. of VLDB*, pages 564–575, 2004.
4. O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. J. Shekita, B. Sznajder, S. Yogev, and S. Yogev. Beyond basic faceted search. In *Proc. of WSDM*, pages 33–44, 2008.
5. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. of ICDE*, pages 431–440, 2002.
6. S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *ACM Trans. Database Syst.*, 31(3):1134–1168, 2006.
7. I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1/2):143–175, 2001.
8. M. Drosou and E. Pitoura. ReDRIVE: result-driven database exploration through recommendations. In *Proc. of CIKM*, pages 1547–1552, 2011.
9. G. J. Fakas. A novel keyword search paradigm in relational databases: Object summaries. *Data Knowl. Eng.*, 70(2):208–229, 2011.
10. H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *Proc. of SIGMOD*, pages 305–316, 2007.
11. V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *Proc. of VLDB*, pages 850–861, 2003.
12. V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proc. of VLDB*, pages 670–681, 2002.
13. V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proc. of VLDB*, pages 505–516, 2005.
14. G. Koutrika, Z. M. Zadeh, and H. Garcia-Molina. Data clouds: summarizing keyword search results over structured data. In *Proc. of EDBT*, pages 391–402, 2009.
15. Z. Peng, J. Zhang, S. Wang, and L. Qin. Treecluster: Clustering results of keyword search over databases. In J. Yu, M. Kitsuregawa, and H. Leong, editors, *Advances in Web-Age Information Management*, volume 4016 of *Lecture Notes in Computer Science*, pages 385–396. Springer Berlin Heidelberg, 2006.
16. S. B. Roy, H. Wang, G. Das, U. Nambiar, M. K. Mohania, and M. K. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *Proc. of CIKM*, pages 13–22, 2008.
17. A. Silberschatz, A. Tuzhilin, and A. Tuzhilin. On subjective measures of interestingness in knowledge discovery. In *Proc. of KDD*, pages 275–281, 1995.
18. A. Simitsis, G. Koutrika, and Y. E. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *VLDB J.*, 17(1):117–149, 2008.
19. K. Stefanidis, M. Drosou, and E. Pitoura. *You May Also Like* results in relational databases. In *Proc. of PersDB*, pages 37–42, 2009.
20. K. Stefanidis, M. Drosou, and E. Pitoura. PerK: personalized keyword search in relational databases through preferences. In *Proc. of EDBT*, pages 585–596, 2010.
21. O. Zamir and O. Etzioni. Grouper: A dynamic clustering interface to web search results. *Computer Networks*, 31(11-16):1361–1374, 1999.