Noname manuscript No.
(will be inserted by the editor)

XPloreRank: Exploring XML Data via You May Also Like Queries

Mehdi Naseriparsa · Chengfei Liu · Md. Saiful Islam · Rui Zhou

Received: date / Accepted: date

Abstract In many cases, users are not familiar with their exact information needs while searching complicated data sources. This lack of understanding may cause the users to feel dissatisfaction when the system retrieves insufficient results after they issue queries. However, using their original query results, we may recommend additional queries which are highly relevant to the original query. This paper presents XPloreRank to recommend top-l highly relevant keyword queries called "*You May Also Like*" (YMAL) queries to the users in XML keyword search. To generate such queries, we firstly analyze the original keyword query results content and construct a weighted co-occurring keyword graph. Then, we generate the YMAL queries by traversing the co-occurring keyword graph and rank them based on the following correlation aspects: (a) *external correlation*, which measures the similarity of the YMAL query to the original query and (b) *internal correlation*, which measures the capability of the YMAL query keywords in producing meaningful results with respect to the data source. Due to the complexity of generating YMAL queries, we propose a novel A* search-based technique to generate top-l YMAL queries efficiently. We also present a greedy-based approximation for it to improve the performance further. Extensive experiments verify the effectiveness and efficiency of our approach.

Keywords XML keyword search · Data exploration · Recommendations

Mehdi Naseriparsa
Swinburne University of Technology, Melbourne, Australia
E-mail: {mnaseriparsa@swin.edu.au}

Chengfei Liu
Swinburne University of Technology, Melbourne, Australia
E-mail: {cliu@swin.edu.au}

Md. Saiful Islam
Griffith University, Gold Coast, Australia
E-mail: {saiful.islam@griffith.edu.au}

Rui Zhou
Swinburne University of Technology, Melbourne, Australia
E-mail: {rzhou@swin.edu.au}

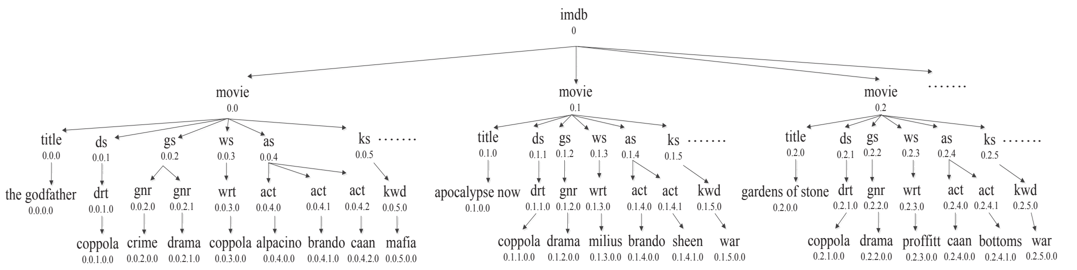


Fig. 1: A part of IMDB dataset

Table 1: Drama movies directed by Coppola and acted by Al Pacino

$Actor1$	$Director$	$Genre_1$	$Writer_1$	$Writer_2$	$Title$	$Genre_2$	$Actor_1$	$Actor_2$
alpacino	**coppola**	**drama**	francis coppola	mario puzo	the godfather	crime	marlon brando	richard castellano
alpacino	**coppola**	**drama**	francis coppola	mario puzo	the godfather II	crime	robert duvall	robert de niro

$Actor1$	$Director$	$Genre_1$	Kw_1	Kw_2	Kw_3	Kw_4	Kw_5	Kw_6
alpacino	**coppola**	**drama**	mafia	wedding	lawyer	violence	organized crime	patriarch
alpacino	**coppola**	**drama**	cuba	new york	business	nevada	escape	shotgun

1 Introduction

Keyword search provides a convenient tool for the users to search complex databases without being familiar with the underlying data source structure. Although keyword search capability provides an easy environment for searching the databases, the users should use exact keywords to retrieve their desired results. However, users may not always be fully familiar with the underlying data source content or they may have a very vague idea about their information needs. The users may issue queries which are insufficient to produce the desired results. This could lead the users to feel dissatisfaction over the system. In this case, a keyword search system equipped with exploratory option for assisting the users to explore additional information via "You May Also Like" (YMAL) queries may become useful and improve the system's overall usability [13], [14].

Motivation. Consider a user keyword query $q_0 = \{alpacino, coppola, drama\}$ issued in the Internet Movie Database (IMDB) as shown in Fig. 1. If we carefully check the IMDB schema, we can infer that *alpacino* is the name of an "actor", *coppola* is the name of a "director" or "writer", and *drama* is a movie "genre". This implies that the user's search intention is to retrieve all *drama* movies directed or written by *coppola* in which *alpacino* is an actor. From Fig. 1, we see that the result for q_0 is the "$Godfather$" movie. However, the user may also be interested in other *drama* movies directed or written by *coppola* in which *alpacino* is *not* an actor but others like *marlon brando* or *james caan*, or *crime* movies directed or written by *coppola* in which *alpacino* is an actor, etc. Currently, these movies such as "*Apocalypse Now*" and "*Gardens of Stone*" (see Fig. 1) are not a part of the original results but they are highly relevant. These additional results can be explored via YMAL queries.

There is a wide range of research conducted on the user query refinement to assist the users when their original queries do not retrieve expected or quality results. When the original query retrieves insufficient or empty results, query relaxation techniques modify or remove some constraints in the original query to produce additional or non-empty results [7], [12]. Conversely, when the original query retrieves many results, query reformulation techniques restrict the results by tightening or introducing new constraints on the original query [20], [15]. The above works focus on modifying the query to restrict or relax the original results. However, sometimes users need to be guided about the similar contents to their original results in terms of exploratory search in the data source. Drosou and Pitoura [8] studied data exploration via recommending additional items in relational databases with structured queries (SQL). However, the problem of exploration in semi-structured data like XML with keyword queries has not been studied in the literature yet.

In connection with XML keyword search, we argue that the co-occurring keywords in the original result content provide a good hint for the YMAL queries since these keywords are highly correlated to the original query results. Therefore, to generate the YMAL queries, we can first analyze the original results content to discover the co-occurring keywords in the original result set and put them into a keyword pool. Then, we can generate YMAL queries by selecting the keywords from this keyword pool. Since we can select any subset of the keywords from the keyword pool, there will be an exponential number of queries.

For example, consider the keyword query $q_0 = \{alpacino,\ coppola,\ drama\}$ again. Table 1 shows the co-occurring keywords within the result content of q_0. We can construct the co-occurring keyword pool for q_0 as follows: $\{alpacino, coppola, drama, crime, puzo, english, italian, latin, corleone, brando, caan, deniro, mafia, wedding, lawyer, violence\}$. A straightforward approach would generate $(2^{16} - 1)$ YMAL queries from the above keyword pool. Some of these queries are as follows: $q_1 = \{brando, coppola,\ drama\}$, $q_2 = \{caan,\ coppola, drama\}$, $q_3 = \{deniro,\ coppola, drama\}$, etc. However, not all the keywords in the co-occurring keyword pool are similar to the original query keywords to the same degree. Furthermore, the selected keywords should be relevant to each other to produce meaningful results with respect to the data source. To score the YMAL queries, we propose two kind of correlation degrees: (a) *external correlation* degree which is the correlation degree between the original keyword query and the YMAL keyword query, (b) *internal correlation* degree which is the correlation degree between the keywords of the YMAL query with respect to data source. The external correlation degree guarantees that the YMAL query is highly relevant to the original query while the internal correlation degree guarantees that the YMAL query will generate meaningful results in the data source. As there are exponential number of queries and we have to optimize their correlation scores, the straightforward approach is inefficient.

As the straightforward approach is computationally expensive, we propose to construct a weighted co-occurring keyword graph from the strongly correlated keywords in the co-occurring keyword pool. Then, we propose a novel A^* search-based technique to traverse the graph and generate the optimal top-l YMAL queries with maximum correlation scores efficiently. We also propose a greedy-based approximation for the A* search-based technique with an error guarantee to quickly extract the suboptimal top-l YMAL queries. We implement these algorithms into a system called *XML Exploration via Ranked YMAL Queries* (XPloreRank) for recommending top-l YMAL queries to the users to explore some interesting and relevant results in addition to their original keyword query results. To be specific, our main contributions are as follows:

1. we propose YMAL queries in XML keyword search and formulate this problem using two correlation scores;
2. we propose a novel technique to extract the optimal top-l YMAL queries from the co-occurring keyword pool by conducting A^* search on the corresponding co-occurring keyword graph;
3. we also propose a greedy-based approximation for the A* search-based technique to quickly extract the suboptimal top-l YMAL queries with an error guarantee, which improves the performance substantially; and
4. finally, we conduct extensive experiments to verify the effectiveness and efficiency of our approach.

The rest of the paper is organized as follows: Section 2 discusses XML keyword search and presents the top-l YMAL queries problem; Section 3 presents our A* search-based technique for generating the optimal top-l YMAL queries based on co-occurring keyword graph; Section 4 presents our greedy-based approximation; In Section 5, we discuss some adaptation for our proposed techniques to vary the YMAL queries size. Section 6 validates our approach experimentally; Section 7 reviews the related work and finally, Section 8 concludes the paper.

2 Background

2.1 Preliminaries

An XML document is modeled as a tree T that contains labeled nodes and a designated root. Each node in the tree T is indexed by a unique identifier called Dewey code, which consists the path from the root to the corresponding node. From Fig. 1, the Dewey code 0.0 refers to a node in the data source that contain information about the *Godfather* movie. A node m_i in the tree T that contains keyword k_i is a match node for k_i. E.g, the match node of the keyword $k_i = mafia$ is $m_i = [0.0.5.0.0]$.

Keyword Query and Subtree Result. In XML data, a keyword query q consists of a set of keywords $\{k_1, k_2, ..., k_n\}$. A result $r = (v_{slca}, \{m_1, m_2, ..., m_n\})$ for q is a subtree in T which contains all keywords $k_i \in q$. The node v_{slca} is the smallest lowest common ancestor (SLCA) of the nodes $\{m_1, m_2, ..., m_n\}$ denoted by $v_{slca} \preceq_a \{m_i, \forall i \in [1-n]\}$ because they contain the Dewey Code of the v_{slca} in their prefixes. Here, we used the popular SLCA semantics [22, 21] to retrieve the XML keyword search results; however, our solutions are independent of this semantics and other LCA-based semantics [11, 23, 24, 4] can be used instead.

Subtree Result Leaf Nodes: For a result r, a result leaf node m' is a node in T that have no child nodes and $v_{slca} \preceq_a m'$. E.g. for $r = ([0.0], \{[0.0.1.0.0], [0.0.2.1.0], [0.0.4.0.0]\})$ in Fig. 1, some of the subtree result leaf nodes are as follows: $m'_1 = [0.0.4.1.0]$, $m'_2 = [0.0.4.2.0]$ and $m'_3 = [0.0.5.0.0]$.

2.2 Problem Statement

In order to explore the data source via YMAL queries to retrieve additional results which are meaningful to the user, we have to generate queries that are highly correlated to the original query and the data source. The YMAL queries produce results that may not be a part of original query results; however, they are highly correlated to the original query results. To generate such queries, we propose to extract the co-occurring keywords within the original query result content and to use them as the possible interesting keywords to generate the YMAL queries.

Definition 1 Co-occurring Keyword. Given the original query result $r = (v_{slca}, \{m_1, ..., m_n\})$ and a set of result leaf nodes $\{m'_1, m'_2, \dots, m'_l\}$ in T, the co-occurring keywords $\mathcal{K}$ include the corresponding keywords of the result leaf nodes.

After finding the co-occurring keywords, we have to generate YMAL queries, denoted by $\mathcal{Q}$, from them. Assume $\mathcal{K}$ is the co-occurring keyword pool. We define a YMAL query $q \in \mathcal{Q}$ as:

Definition 2 YMAL Query. Given a co-occurring keyword pool $\mathcal{K}$, a YMAL query $q = \{k_1, k_2, ..., k_i\}$ is a subset of $\mathcal{K}$, where $i \in \{1, 2, ..., |\mathcal{K}|\}$.

However when generating the YMAL queries $\mathcal{Q}$ from $\mathcal{K}$, we face combinatorial explosion since the number of such YMAL queries is exponential to the cardinality of $\mathcal{K}$. Therefore, we have to limit the number of these queries to the most promising ones. We propose a scoring scheme that takes two correlation parameters into account: (a) *internal correlation* to ensure that the selected keywords $k_i \in q$ will produce meaningful results with respect to the data source, and (b) *external correlation* to ensure that the YMAL queries are highly similar to the original query q_0.

Assume $\lambda^{in}(q, T)$ and $\lambda^{ex}(q, q_0, T)$ denote the internal and external correlation degrees of a YMAL query $q \in \mathcal{Q}$, respectively. Then, we score the YMAL query q as follows:

$$\lambda(q, T) = \alpha \times \lambda^{in}(q, T) + (1 - \alpha) \times \lambda^{ex}(q, q_0, T) \tag{1}$$

where α is a tuning parameter to tradeoff between the internal and external correlations.

Definition 3 Top-l YMAL Queries. Given a keyword query q_0 on T and its result set $\mathcal{R}$, the top-l YMAL queries $\mathcal{Q}^*$ is a subset of $\mathcal{Q}$ which maximize the scoring function given in Eq. 1.

Note that, we do not specifically focus on diversifying the YMAL queries. We can enforce diversity by considering the dissimilarities between selected top-l YMAL queries. However, a better diversification will require estimating the query results and discovering a good result coverage that is out of the scope of this paper.

3 Our Approach

To generate top-l YMAL queries from $\mathcal{K}$, a straightforward approach would firstly generate all the subsets from the co-occurring keyword pool $\mathcal{K}$. Then, it would score these queries in terms of their correlation degrees and retrieves the top-l highly correlated YMAL queries $\mathcal{Q}^*$. However, the complexity of this approach is $\mathcal{O}(2^{|\mathcal{K}|})$, which is inefficient. Moreover, not all the subsets are appropriate for presenting to the user as YMAL queries. For instance, some YMAL queries may produce many results that are not relevant because the size of query is small and it generates loose results. Conversely, when the size of the query is big, it may tighten the results and cause non-answer query problem. That's because if we add more keywords to a YMAL query, it may decrease its correlation score. Therefore, to avoid selecting the queries which produce loose results or the queries which produce non-answers, we propose to generate the YMAL queries limited to a fixed size which can be set adaptively which is described in Section 5. Let the limit be the original query size $|q_0|$ for now. To retrieve the top-l YMAL queries in XML, we propose to construct a co-occurring keyword graph $\mathcal{G}$ and traverse this graph to measure the correlation score of the promising YMAL queries only instead of generating all possible queries from $\mathcal{K}$. The general framework of our approach is depicted in Fig. 2.

3.1 Co-occurring Keyword Graph

The co-occurring keywords $k \in \mathcal{K}$ has a correlation degree to each other and to the original keywords. Thus, these keywords are connected together via these correlation degrees and they build the co-occurring keywords graph $\mathcal{G}$ as follows:

Definition 4 Co-occurring Keyword Graph. Given the co-occurring keywords $\mathcal{K} = \{k_1, k_2, ..., k_{|\mathcal{K}|}\}$ and the correlation between keywords $cor(k_i, k_j)$, then the co-occurring keyword graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is an undirected weighted graph which:

- $\mathcal{V} = \mathcal{K}$ and each $k_i \in \mathcal{V}$ has a correlation feature vector $k_i.vect = \{f_1, f_2, ..., f_n\}$ that reflects its correlation degree to the original keywords.
- $\mathcal{E}$ is a set of edges that connect the keywords in $\mathcal{G}$. There is a weight on each connecting edge such that $edge.weight = \{cor(k_i, k_j), k_i, k_j \in \mathcal{V}\}$. Two keywords in $\mathcal{G}$ have connecting edge if $cor(k_i, k_j) \geq \eta$.

3.1.1 Keyword Pairwise Correlation

After building the co-occur- ring keyword pool $\mathcal{K}$, we measure the correlation degree of each keyword pair $k_i, k_j \in \mathcal{K}$. The pairwise correlation measure determines how two keywords are relevant to each other and may generate a YMAL query with meaningful results. We use the semantic metric *Normalized Google Distance* [6] $NGD(k_i, k_j) = max\{\log|\mathcal{R}(k_i)|, \log|\mathcal{R}(k_j)|\} - \log|\mathcal{R}\ (k_i, k_j)|/\log N - min$ $\{\log|\mathcal{R}(k_i)|, \log|\mathcal{R}(k_j)|\}$ to compute the correlation degree between two keywords k_i and k_j using the Web data source W as follows:

$$cor(k_i, k_j, W) = e^{-2NGD(k_i, k_j)} \tag{2}$$

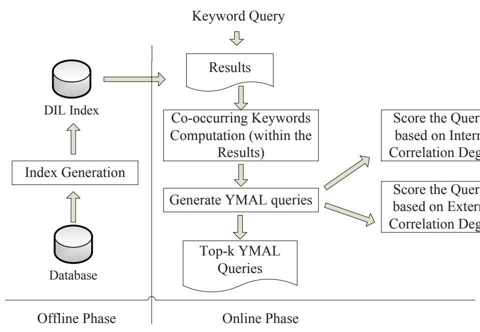

Keyword Query
Results
DIL Index
Co-occurring Keywords Computation (within the Results)
Score the Queries based on Internal Correlation Degree
Index Generation
Generate YMAL queries
Score the Queries based on External Correlation Degree
Database
Top-k YMAL Queries
Offline Phase
Online Phase

Fig. 2: XPloreRank approach architecture

where N is the total number of web documents, $\mathcal{R}(k_i)$ and $\mathcal{R}(k_j)$ return the results in W which contain k_i and k_j, respectively, and $\mathcal{R}(k_i, k_j)$ returns the results which contain both k_i and k_j. However, this metric computes the correlation degree based on the cardinality of the results that are retrieved by a particular search engine on an external data source. To measure the correlation degree w.r.t. the local data source, we use the Jaccard coefficient as:

$$cor(k_i, k_j, T) = \frac{|\mathcal{R}(k_i) \cap \mathcal{R}(k_j)|}{|\mathcal{R}(k_i) \cup \mathcal{R}(k_j)|} \tag{3}$$

where $\mathcal{R}(k_i)$ and $\mathcal{R}(k_j)$ return the results in T which contain k_i and k_j respectively. Note that in XML data, we use the number of SLCA nodes that contain k_i and k_j for the numerator and the sum of the inverted list sizes of k_i and k_j for the denominator. Finally, we combine the above equations to compute the final correlation degree between k_i and k_j as follows:

$$cor(k_i, k_j) = \gamma \times cor(k_i, k_j, W) + (1 - \gamma) \times cor(k_i, k_j, T) \tag{4}$$

where γ is a tuning parameter for trading off between the correlation measures of the Web W and the local data source T. However, our method for top-l YMAL queries is independent from the keywords correlation computation. Thus, any kind of correlation metrics can be used instead.

3.1.2 Keyword Correlation to the Original Query

To generate similar YMAL queries from the original query results, the co-occurring keywords should be as close as possible to the original query keywords. Therefore, we have to measure the relevance degree of the co-occurring keywords to the original query keywords. Since a keyword query consists of several keywords, we measure the correlation of each keyword $k \in \mathcal{K}$ with all the original keywords $\forall k_i \in q_0, i \in [1, n]$ using the Eq. 4. Therefore each keyword $k \in \mathcal{K}$, has a set of correlation scores to the original query keywords like a vector as given as follows:

Definition 5 Correlation Feature Vector. Given a co-occurring keyword $k \in \mathcal{K}$, the correlation feature vector $k.vect = (f_1, f_2, ..., f_n)$ is a set of entries such that: $f_i = cor(k, k_i \in q_0), 1 \leq i \leq n$.

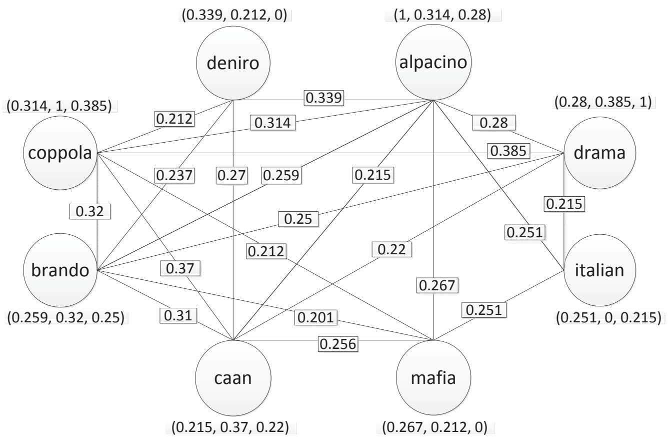


Fig. 3: Co-occurring keyword graph

Example 1 Consider the keyword *caan* and the following correlation degrees: $cor(caan, alpacino) = 0.74$, $cor(caan, copp\ ola) = 0.367$, $cor(caan, drama) = 0.319$. Then $caan.vect = (0.74, 0.367, 0.319)$.

Example 2 Consider the original keyword query $q_0 = \{alpacino, coppola, drama\}$ and the given co-occurring keyword pool $\mathcal{K} = \{alpacino, coppola, drama, italian, brando, caan, mafia, deniro\}$. The co-occurring keyword graph $\mathcal{G}$ with $\eta \geq 0.2$ is presented in Fig. 3.

3.2 Scoring

After constructing the co-occurring graph $\mathcal{G}$, we generate the YMAL queries by traversing $\mathcal{G}$. However, the number of generated queries is enormous. Furthermore, not all the queries from traversing $\mathcal{G}$ would produce meaningful results in the data source. Also, not all the queries are similar to the original query to the same degree. Therefore, we have to score the queries based on their ability to produce meaningful results with respect to data source and their similarity degree to the original query. We use the query internal correlation to measure the YMAL queries ability to produce meaningful results and the query external correlation to measure their similarity degree to the original query.

3.2.1 Internal Correlation

To compute the internal correlation degree of a YMAL query $q \in \mathcal{Q}$, we have to find the correlation degree between its constituent keywords. Therefore, all the pairwise correlation between the keywords should be measured as:

$$\lambda^{in}(q, T) = \begin{cases} 0, & \text{if } |q| = 1 \\ \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} cor(k_i, k_j \in q)}{|q|}, & \text{if } |q| \geq 2 \end{cases} \tag{5}$$

Consider there are n keywords in the YMAL query q. Thus, there are $C(n, 2)$ keyword pairs in q. Each pair has a correlation degree; therefore, we use the normalized summation of the keyword pairs correlation degrees to measure the YMAL query internal correlation.

3.2.2 External Correlation

Since a YMAL query $q \in \mathcal{Q}$ is a set of co-occurring keywords, the query correlation vector is produced by getting the average value for each vector entry of each keyword $k_i \in q$ as follows: $q.vect = \{avg((f_1^1, f_1^2, ..., f_1^n), ..., (f_n^1, f_n^2, ..., f_n^n))\}$ where f_i^j denotes the jth entry for $k_i \in q$ in the correlation feature vector. In order to measure the correlation of a YMAL query q to the original query q_0, we use the summation of the YMAL query correlation vector elements to compute the total value. Thus, for the YMAL query q, the external correlation degree $\lambda^{ex}(q, q_0, T)$ is measured as follows:

$$\lambda^{ex}(q, q_0, T) = \frac{\sum_{i=1}^{n} q.vect.f_i}{|q|} \tag{6}$$

Finally, both $\lambda^{in}(q, T)$ and $\lambda^{ex}(q, q_0, T)$ contribute to the $\lambda(q, T)$ that is presented in Eq. 1.

Example 3 Assume $\alpha = 0.5$. Then the internal and external correlation degrees of the YMAL query $q_1 = \{brando, coppola, drama\}$ generated from $\mathcal{K}$ of Example 2 are as follows: $cor(brando, coppola) = 0.32$, $cor(brando, drama) = 0.25$, $cor(coppola, drama) = 0.385$, According to Eq. 5, $\lambda^{in}(q_1, T) = \frac{0.32+0.25+0.385}{3} = 0.318$, $q_1.vect = avg((0.259, 0.32, 0.25), (0.314, 1, 0.385), (0.28, 0.385, 1))$, $q_1.vect = (0.284, 0.568, 0.532)$. Then according to Eq. 6, $\lambda^{ex}(q_1, q_0, T) = \frac{0.284+0.568+0.532}{3} = 0.461$. Finally, from Eq. 1 the total correlation degree is $\lambda(q_1, T) = 0.5 \times 0.318 + 0.5 \times 0.461 = 0.39$.

Theorem 1 *Generating the YMAL query $q \in \mathcal{Q}$ from the co-occurring keyword graph $\mathcal{G}$ with the maximum $\lambda(q, T)$ and with the constraint $|q| = |q_0|$ is an NP-hard problem.*

Proof We reduce the well-known NP-hard problem of n Densest Subgraph to our problem. Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and number n, the n Densest Subgraph problem finds a subgraph $\mathcal{G}_s(\mathcal{V}_s, \mathcal{E}_s)$ with n vertices such that $|\mathcal{V}_s| = n$ and the density $\frac{\sum_{u,v \in \mathcal{E}_s} w(u,v)}{|\mathcal{V}_s|}$ is maximized. From this, we construct an instance of our problem, consisting of graph $\mathcal{G}$, parameters $|q_0| = n$ and $\alpha = 1$. We show that the instance of the Densest Subgraph problem is a YES-instance iff the corresponding instance of our problem is a YES-instance. Given a solution q for our problem, q must contain $|q_0|$ keywords. Since $\lambda^{in}(q, T)$ should be the maximum, q is a subgraph of size $|q_0|$ from $\mathcal{G}$ that has the best density.

3.3 Query Processing: A* Search

The brute force approach finds the optimal solution $\mathcal{Q}^*$; however, it generates all the possible queries that makes it impractically expensive. Therefore to improve the performance and not to generate all queries $\mathcal{Q}$, we provide an efficient approach using A* search to generate the top-l YMAL queries. In this approach, we traverse the co-occurring keywords graph $\mathcal{G}$ to find the keywords that could generate

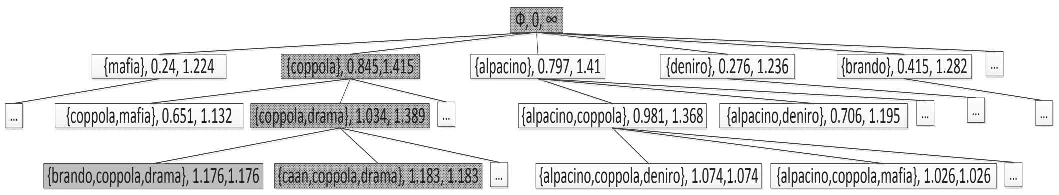

Φ, 0, ∞
{mafia}, 0.24, 1.224
{coppola}, 0.845,1.415
{alpacino}, 0.797, 1.41
{deniro}, 0.276, 1.236
{brando}, 0.415, 1.282
...
...
{coppola,mafia}, 0.651, 1.132
{coppola,drama}, 1.034, 1.389
...
{alpacino,coppola}, 0.981, 1.368
{alpacino,deniro}, 0.706, 1.195
...
...
...
{brando,coppola,drama}, 1.176,1.176
{caan,coppola,drama}, 1.183, 1.183
...
{alpacino,coppola,deniro}, 1.074,1.074
{alpacino,coppola,mafia}, 1.026,1.026
...

Fig. 4: A part of A* search process

promising results. To traverse $\mathcal{G}$, we use an upper bound estimation to predict the score of a YMAL query during the search process. This upper bound is updated and becomes tighter in each step. We exploit this upper bound to prune many non-promising computations that would accelerate the search process. To generate the top-l YMAL queries, we use a max heap $\mathcal{H}$ to store the partial results and perform the search on co-occurring keywords $\mathcal{G}.\mathcal{V}$ space, and finally, find YMAL queries with maximum scores.

Assume that q^p is a partial YMAL query (i.e., $|q^p| < |q_0|$), q^u is the unseen part of the YMAL query q such that $q = q^p \cup q^u$ and $|q^u| = |q| - |q^p|$, $\overline{\lambda}^{in} = max\{cor(k_i, k_j), \forall k_i, k_j \in \mathcal{K} \setminus q^p\}$ and $d = C(|q_0|, 2) - C(|q^p|, 2)$. Then, the upper bound of the internal correlation degree of q, denoted by $\overline{\lambda}^{in}(q, T)$, becomes as follows:

$$\overline{\lambda}^{in}(q,T) = \begin{cases} \frac{|q_0| \times \overline{\lambda}^{in}}{|q|}, & \text{if } |q^p| = 1 \\ \frac{\lambda^{in}(q^p,T) + d \times \overline{\lambda}^{in}}{|q|}, & \text{if } |q^p| > 1 \end{cases} \tag{7}$$

Again, assume that $q^u.\overline{vect} = max\{k_i.vect, \forall k_i \in \mathcal{K} \setminus q^p\}$. Then, the upper bound of the correlation feature vector of the YMAL query q becomes: $q.\overline{vect} = avg(q^p.vect, (|q_0| - |q^p|) \times q^u.\overline{vect})$. We get the upper bound of the external correlation degree of q, denoted by $\overline{\lambda}^{ex}(q, q_0, T)$, as given as follows:

$$\overline{\lambda}^{ex}(q, q_0, T) = \frac{\sum_{i=1}^{n} q.\overline{vect}.f_i}{|q|} \tag{8}$$

Lemma 1 *The upper bound of the correlation score of q $\lambda(q, T)$, denoted by $\overline{\lambda}(q, T)$, is as follows:*

$$\overline{\lambda}(q,T) = \alpha \times \overline{\lambda}^{in}(q,T) + (1 - \alpha) \times \overline{\lambda}^{ex}(q, q_0, T) \tag{9}$$

The upper bound estimation of the score of q never underestimates the real score of q; therefore, it is admissible.

Lemma 2 *Given $\lambda(q^*, T)$, $\overline{\lambda}(q_1, T)$ such that $\forall q_2 \in \mathcal{Q}$, $\overline{\lambda}(q_1, T) \geq \overline{\lambda}(q_2, T)$ where $|q^*|, |q_1|, |q_2| = |q_0|$. The stop condition $\lambda(q^*, T) \geq \overline{\lambda}(q_1, T)$ retrieves the optimal q^* if $\overline{\lambda}(q_1, T)$ is admissible.*

Proof Since our estimation of $\overline{\lambda}(q_1, T)$ is admissible, we have $\overline{\lambda}(q_1, T) \geq \lambda(q_1, T)$. We know that $\overline{\lambda}(q_1, T)$ is the maximum value among other estimations. Therefore using the admissibility, $\forall q_2 \in \mathcal{Q}, \lambda(q_2, T) \leq \overline{\lambda}(q_1, T) \leq \lambda(q^*, T)$; therefore, q^* is the optimal YMAL query. Hence, the lemma.

Algorithm 1: A* Search-based Technique

```
   Input  : $q_0$, $\mathcal{S}$, $\alpha$, $l$
   Output: Top-$l$ YMAL Queries $\mathcal{Q}^*$
1  $\mathcal{Q}^* \leftarrow \emptyset$;$\mathcal{H} \leftarrow \emptyset$;                                   // initialization
2  $\mathcal{K} \leftarrow findKeywords(q_0, \mathcal{S})$;                       // co-occurring keywords
3  $\mathcal{G} \leftarrow generateGraph(\mathcal{K})$;                              // keyword graph
4  $\mathcal{V}' \leftarrow sort(\mathcal{G}.\mathcal{V}, \lambda^{ex})$; $\mathcal{E}' \leftarrow sort(\forall k_i, k_j \in \mathcal{G}.\mathcal{V}, cor(k_i, k_j))$;
5  while $k_i = getNext(\mathcal{G}.\mathcal{V}) \neq \emptyset$ do
6      $e.q \leftarrow k_i$; $e.\lambda \leftarrow (1-\alpha) \times \lambda^{ex}(k_i, q_0, T)$;
7      $e.\overline{\lambda} \leftarrow measureUpperBound(e, q_0, \mathcal{G}, \mathcal{V}', \mathcal{E}')$;          // Eq. 9
8      $\mathcal{H}.push(e)$;                                          // insert $e$ into $\mathcal{H}$
9  $\lambda^{min} \leftarrow 0$; $\mathcal{Q}^* \leftarrow \emptyset$;                                  // initialization
10 while $\mathcal{H} \neq \emptyset$ do
11     $e \leftarrow generateQuery(q_0, \mathcal{H}, \mathcal{G}, \lambda^{min}, \mathcal{V}', \mathcal{E}')$;
12     if $e \neq null$ then
13         if $|\mathcal{Q}^*| = l$ then
14             $\mathcal{Q}^* \leftarrow update(\mathcal{Q}^*, e.q)$;                  // update top-$l$ with $e.q$
15             $\lambda^{min} \leftarrow minScore(\mathcal{Q}^*)$;                         // update $\lambda^{min}$
16         else
17             $\mathcal{Q}^* \leftarrow \mathcal{Q}^* \cup e.q$;                                // add query to top-$l$
18     else
19         break;                                   // top-$l$ YMAL queries computed
20 return $\mathcal{Q}^*$;
```

The core of the A* search-based technique is the search procedure that carries out an informed search on the co-occurring keyword graph $\mathcal{G}$ to find the optimal YMAL query q^*. Since the informed search is costly and impractical, we can reuse some computations for generating top-l YMAL queries during the search on $\mathcal{G}$. For computing top-l YMAL queries $\mathcal{Q}^* = \{q_1, q_2, ..., q_l\}$, the partial YMAL queries computations in the max heap $\mathcal{H}$ for computing $q_i \in \mathcal{Q}^*$ can be reused when computing q_{i+1}.

Example 4 Consider the original query $q_0 = \{alpacino, coppola, drama\}$ and the keyword pool $\mathcal{K}$ presented in Example 2 and $\alpha = 0.5$. A part of the A* search process is shown in Fig. 4. At first, the top entry $(\emptyset, 0, \infty)$ is popped from the heap $\mathcal{H}$, and 8 new entries are pushed ($|\mathcal{K}| = 8$). The entry with the partial query $q_1^p = \{coppola\}$ is expanded first because it has the best upper bound $\overline{\lambda}(q_1, T) = 0.472$. After that, the entry with the partial query $q_2^p = \{alpacino\}$ is expanded with the upper bound $\overline{\lambda}(q_2, T) = 0.47$. Note that the score of a parent partial query may be larger than the score of its child partial query because the score function is not monotone and adding some less correlated keywords may decrease the partial query score. For example, in the partial query $q_y^p = \{coppola, mafia\}$ the score decreased from 0.282 to 0.217. In the expanded entry with the partial query $q_1^p = \{coppola, drama\}$, the upper bound score is produced as follows: $\overline{\lambda}^{in} = \frac{0.385+2\times0.37}{3} = 0.375$, $\overline{vect} = (0.531, 0.567, 0.555)$, $\overline{\lambda}^{ex} = 0.551$, $\overline{\lambda}(q_1, T) = 0.5 \times 0.375 + 0.5 \times 0.551 = 0.463$. The entry $(\{caan, coppola, drama\}, 0.394, 0.394)$ contains the top-1 YMAL query q_1. For computing the top-2 YMAL query q_2, we can use the current computations and pop $(\{brando, coppola, drama\}, 0.392, 0.392)$ from $\mathcal{H}$.

Algorithm 1 presents the framework for A* approach. We initialize a max heap $\mathcal{H}$ to store the A* search information in line 1. In lines 2-3, we construct

Algorithm 2: generateQuery Procedure for A* Search-based Technique

```
Input  : $q_0$, $\mathcal{H}$, $\mathcal{G}$, $\lambda^{min}$, $\mathcal{V}'$, $\mathcal{E}'$
Output: $e$ The element containing YMAL Query
1  while $\mathcal{H} \neq \emptyset$ do
2      $e \leftarrow \mathcal{H}.pop()$;
3      if $e.\overline{\lambda} < \lambda^{min}$ then
4          return $null$;                                   // as per Lemma 2
5      if $|e.q| < |q_0|$ then
6          while $k_i = getNext(\mathcal{G}.\mathcal{V}) \neq \emptyset$ and $k_i \cap e.q = \emptyset$ do
7              $e'.q \leftarrow e.q \cup k_i$;                 // add keyword $k_i$ to the query
8              $e'.\lambda \leftarrow \alpha \times \lambda^{in}(e'.q, T) + (1-\alpha) \times \lambda^{ex}(e'.q, q_0, T)$;
9              $e'.\overline{\lambda} \leftarrow measureUpperBound(e', q_0, \mathcal{G}, \mathcal{V}', \mathcal{E}')$;
10             $\mathcal{H}.push(e')$;                          // insert $e'$ into $\mathcal{H}$
11     else
12         $\lambda^{min} \leftarrow e.\lambda$;
13         $q \leftarrow e.q$;
14         return $e$ ;                                     // return the YMAL query
```

Algorithm 3: measureUpperBound

```
Input  : $e$, $q_0$, $\mathcal{G}$, $\mathcal{V}'$, $\mathcal{E}'$
Output: score upper bound $\overline{\lambda}$
1  if $|e.q| = |q_0|$ then
2      return $e.\lambda$;
3  $\overline{\lambda}^{ex}, \overline{\lambda}^{in} \leftarrow 0$;
4  $mark(\mathcal{V}', e.q)$;
5  $mark(\mathcal{E}', e.q)$;
6  $d \leftarrow C(|q_0|, 2) - C(|e.q|, 2)$;
7  if $|e.q| = 1$ then
8      $\overline{\lambda}^{in} \leftarrow \sum top(\mathcal{E}'.cor, |q_0|)$;
9  else
10     $\overline{\lambda}^{in} \leftarrow \lambda^{in}(e.q, T) + \frac{\sum top(\mathcal{E}'.cor, d)}{d}$;
11 $vect \leftarrow avg(top(\mathcal{V}'.vect, |q_0| - |e.q|))$;
12 $\overline{\lambda}^{ex} \leftarrow \frac{\sum_{i=1}^{n} avg(e.q.vect, vect).e_i}{|e.q|}$;
13 $\overline{\lambda} = \alpha \times \overline{\lambda}^{in} + (1-\alpha) \times \overline{\lambda}^{ex}$;
14 return $\overline{\lambda}$
```

the co-occurring keywords $\mathcal{K}$ and build the $\mathcal{G}$. Each entry $e \in \mathcal{H}$ has the following information: (a) $e.q$ which is the partial YMAL query, (b) $e.\lambda$ which is the correlation score of $e.q$, and (c) $e.\overline{\lambda}$ which is the upper bound estimation of the correlation score. In line 4, we sort the vertices and edges of $\mathcal{G}$ based on their correlations respectively and store them into $\mathcal{V}'$ and $\mathcal{E}'$ for measuring the upper bound. In lines 5-8, we initialize the $\mathcal{H}$ with the possible elements of $\mathcal{G}$. In line 11, we invoke the procedure *generateQuery* to find the current best q. In lines 13-15, if $|\mathcal{Q}^*| = l$, we update $\mathcal{Q}^*$ with the query $e.q$ and update the threshold score λ^{min}. Otherwise, we add the query $e.q$ to the top-l YMAL query list $\mathcal{Q}^*$ as pseudocoded in line 17.

The procedure *generateQuery* in line 11 of Algorithm 1 is presented in Algorithm 2. In lines 1-2, if $\mathcal{H} \neq \emptyset$, we pop the top element of $\mathcal{H}$ and store it into e. In lines 3-4, if $e.\overline{\lambda} < \lambda^{min}$, we stop our search since the stop condition is satisfied according to lemma 2. If $|e.q| < |q_0|$, we start to expand the entry e because we have to add more keywords to $e.q$ as pseudocoded in line 5. In line 6, we read the keywords that are not in $e.q$. Then in line 7, we add a keyword $k_i \in \mathcal{G}.\mathcal{V}$ to $e.q$ and generate the new expanded partial query $e'.q$. In lines 8-9, we compute the correlation score of the expanded query $e'.\lambda$ and its upper bound estimated score $e'.\overline{\lambda}$. In lines 10, we push the expanded element into $\mathcal{H}$ for the next iterations. In line 14, we return e for procedure *generateQuery*.

The procedure *measureUpperBound* is pseudocoded in Algorithm 3. In lines 1-2, if $|e.q| = |q_0|$ we return the total score of the entry $e.\lambda$ as the upper bound because the entry contains the ultimate YMAL query. In lines 4-5, we mark the vertices and edges that are a part of the current partial YMAL query $e.q$ (we mark these vertices and edges in order to ignore them for measuring the upper bound). In line 6, we compute d which is the number of keyword correlation pairs difference between the YMAL partial query $e.q$ and the original query q_0. In line 7-8, if $|e.q| = 1$, we compute the summation of top $|q_0|$ elements of $\mathcal{E}'.cor$ (the function $top(\mathcal{E}'.cor, |q_0|)$ returns the top $|q_0|$ correlation scores). Otherwise in line 10, we add λ^{in} to the summation of top d elements of $\mathcal{E}'.cor$ (using the summation of top elements instead of using only the top element would produce a tighter upper bound which improves the performance of the A* search). In line 11, we compute the average of top $(|q_0| - |e.q|)$ vectors of $\mathcal{V}'.vect$. Then in line 12, we compute $\overline{\lambda}^{ex}$. In line 13, we compute the total upper bound score $\overline{\lambda}$.

4 Approximation

The A* search-based technique conducts an informed search on the co-occurring keyword graph $\mathcal{G}$. To conduct an informed search, A* estimates upper bound for the queries during their computation. However, it is difficult to estimate this upper bound which incurs some computational overheads. Moreover, sometimes this upper bound estimation may be loose; therefore, affects the performance of the A* search which makes it highly costly and impractical. Furthermore in exploratory search, users may not need the optimal solution but they need a fast response time. Therefore, we propose a greedy-based approximation to quickly extract the top-l YMAL queries $q \in \mathcal{Q}^*$ in this section. In this approach, we only start the search with the most promising keywords called seed keywords $\mathcal{K}'$. Also, we use an approximation rate θ and compare the real score of each entry with its parent entry. If the score of e is not increased by our approximation rate, we prune this entry because it may not be promising. These prunings lead to improving the performance of the search by ignoring many non-promising computations. Before presenting our method, we discuss some properties of the correlation score function (shown in Eq. 1) that are useful to design the approximate greedy method.

Property 1 The score function $\lambda(q, T)$ is non-negative, i.e., $\lambda(q, T) \geq 0$, $\forall q \in \mathcal{Q}$.

Property 2 The score function $\lambda(q, T)$ is non-monotone since $\exists q_1 \subset q_2 \in \mathcal{Q}$ such that $\lambda(q_1, T) \geq \lambda(q_2, T)$ or $\lambda(q_1, T) < \lambda(q_2, T)$.

Our greedy-based approximation is based on a local search technique. Since the score function $\lambda(q, T)$ is non-monotone, we have to check whether a keyword $k \in q$ increases or decreases the total score. Thus, in this approach we increase the total score of a partial YMAL query $q^p \subseteq q$ by adding a keyword k_i to q^p or excluding a keyword k_j from q^p until this addition or exclusion keeps increasing the total score. The main steps of our greedy-based approximation are as follows.

We start with a set of seed keywords $\mathcal{K}' \subset \mathcal{G}.\mathcal{V}$ with the maximum external correlation to q_0 among other keywords, $\forall k \in \mathcal{K}'$ we create an entry e and push e into the max heap $\mathcal{H}$. Then we repeatedly retrieve $e = \mathcal{H}.pop()$ until $e.\overline{\lambda} < \lambda^{min}$ and do the following: if $|e.q| < |q_0|$ then we check if $\exists k' \in \mathcal{G}.\mathcal{V} \setminus e.q$ such that $\lambda(e.q \cup k', T) > (1+\theta) \times \lambda(e.q, T)$ then $e.q = e.q \cup k'$ and push e into $\mathcal{H}$. If $|e.q| = |q_0|$ then we check if $\exists k' \in e.q$ such that $\lambda(e.q \backslash k', T) > (1+\theta) \times \lambda(e.q, T)$ then $e.q = e.q \backslash k'$ and push e into $\mathcal{H}$, else we return e as the entry that contains YMAL query q. The θ is the approximation rate for our greedy method. We call q is a suboptimal YMAL query if for any YMAL query q_1, q_2 such that $q_1 \subseteq q \subseteq q_2$, $\lambda(q_1, T) \leq \lambda(q, T)$ and $\lambda(q_2, T) \leq \lambda(q, T)$. Here, we define our approximate suboptimal YMAL query.

Definition 6 Approximate Suboptimal YMAL Query. Given the score function $\lambda(q, T)$ and an approximate rate θ, q is an approximate suboptimal YMAL query, if (a) $\forall k_i \in q$, $(1+\theta) \times \lambda(q, T) \geq \lambda(q \backslash k_i, T)$, and (b) $\forall k_j \notin q$, $(1+\theta) \times \lambda(q, T) \geq \lambda(q \cup k_j, T)$.

In the greedy-based approximation, we use the approximation rate θ to check the amount of increase in the total score of a YMAL query and prune those queries that are not promising. Thus, in the greedy approach, we compute both the the upper bound score $\overline{\lambda}$ and the lower bound score $\underline{\lambda}$ of q to compute the approximation rate as follows: (a) $\epsilon = \overline{\lambda}(q, T) - \underline{\lambda}(q, T)$, and (b) $\theta = \frac{\epsilon}{|\mathcal{K}|}$.

Assume that q^p is a partial YMAL query (i.e., $|q^p| < |q_0|$), q^u is the unseen part of the YMAL query q such that $q = q^p \cup q^u$ and $|q^u| = |q| - |q^p|$, $\underline{\lambda}^{in} = min\{cor(k_i, k_j), \forall k_i, k_j \in \mathcal{K} \setminus q^p\}$ and $d = C(|q_0|, 2) - C(|q^p|, 2)$. Then, the lower bound of the internal correlation degree of q, denoted by $\underline{\lambda}^{in}(q, T)$, becomes as follows:

$$\underline{\lambda}^{in}(q, T) = \begin{cases} \frac{|q_0| \times \underline{\lambda}^{in}}{|q|}, & \text{if } |q^p| = 1 \\ \frac{\underline{\lambda}^{in}(q^p, T) + d \times \underline{\lambda}^{in}}{|q|}, & \text{if } |q^p| > 1 \end{cases} \tag{10}$$

Again, assume that $q^u.\underline{vect} = min\{k_i.vect, \forall k_i \in \mathcal{K} \setminus q^p\}$. Then, the lower bound of the correlation feature vector of the YMAL query q becomes: $q.\underline{vect} = avg(q^p.vect, (|q_0| - |q^p|) \times q^u.\underline{vect})$. We get the lower bound of the external correlation degree of q, denoted by $\underline{\lambda}^{ex}(q, q_0, T)$, as given as follows:

$$\underline{\lambda}^{ex}(q, q_0, T) = \frac{\sum_{i=1}^{n} q.\underline{vect}.f_i}{|q|} \tag{11}$$

Lemma 3 *The lower bound of the correlation score of q $\lambda(q, T)$, denoted by $\underline{\lambda}(q, T)$, is as follows:*

$$\underline{\lambda}(q, T) = \alpha \times \underline{\lambda}^{in}(q, T) + (1 - \alpha) \times \underline{\lambda}^{ex}(q, q_0, T) \tag{12}$$

Example 5 Assume the partial YMAL query $q^p = \{coppola,\ drama\}$ presented in Fig. 4. Then the lower bound estimation computed as follows: $\underline{\lambda}^{in} = 0$, $\underline{\lambda}^{in}(q, T) = \frac{0.328+2\times 0}{3} = 0.127$, $q.\underline{vect} = (0.274, 0.347, 0.454), \underline{\lambda}^{ex}(q, q_0, T) = 0.358$. Finally, $\underline{\lambda}(q, T) = 0.5 \times 0.127 + 0.5 \times 0.358 = 0.243$.

Since our lower bound estimation guarantees that any optimal YMAL query score will beat it; therefore, the difference between the upper bound $\overline{\lambda}(q,T)$ and the lower bound $\underline{\lambda}(q,T)$ is the maximum value for the estimation error which we use it as ϵ for the approximation rate.

Theorem 2 *The greedy-based approximation algorithm is a-* $\frac{SUBOPT}{(1+\theta)^{|q_0|}}$ *approximation algorithm for presenting top-l YMAL queries. The algorithm complexity is* $\mathcal{O}(\frac{1}{\epsilon}|\mathcal{K}|^2 \log |\mathcal{K}|)$.

Proof Assume $\lambda(q,T)$ is the score of the approximate suboptimal YMAL query. Since in each iteration the score increases by the factor $(1+\theta)$, then if the algorithm iterations is i we have the following: $(1+\theta)^i \times \lambda(q,T) \geq SUBOPT$, and finally $\lambda(q,T) \geq \frac{SUBOPT}{(1+\theta)^i}$. We present the YMAL queries with the size $|q_0|$; therefore, $\lambda(q,T) \geq \frac{SUBOPT}{(1+\theta)^{|q_0|}}$. For the complexity analysis, assume $\overline{\lambda}$ is the best score for the queries. Thus, its simple to see $|\mathcal{K}| \times \overline{\lambda} \geq SUBOPT$. If the algorithm iteration is i, then in each iteration the score increases by $(1+\frac{\epsilon}{|\mathcal{K}|})$. Then, after i iteration $|\mathcal{K}| \times \overline{\lambda} \geq (1+\frac{\epsilon}{|\mathcal{K}|})^i \times \overline{\lambda}$, $|\mathcal{K}| \geq (1+\frac{\epsilon}{|\mathcal{K}|})^i$, then $i = \mathcal{O}(\frac{1}{\epsilon}|\mathcal{K}| \log |\mathcal{K}|)$. Since the search includes $|\mathcal{K}|$ the complexity becomes $\mathcal{O}(\frac{1}{\epsilon}|\mathcal{K}|^2 \log |\mathcal{K}|)$.

Example 6 Assume $|\mathcal{K}| = 10$, $\lambda(q,T)$ is the score of the approximate suboptimal query q, $\overline{\epsilon} = 1$, and $|q_0| = 3$. The greedy algorithm finds a solution by the following rate: $\lambda(q,T) \geq \frac{SUBOPT}{(1+0.1)^3}$, $\lambda(q,T) \geq 0.75\ SUBOPT$.

4.1 The Algorithm

Algorithm 4 presents the greedy approach. Lines 1-3 is similar to Algorithm 1. In line 4, we sort the vertices and edges of $\mathcal{G}$ based on their correlation score respectively and store them into $\mathcal{V}'$ and $\mathcal{E}'$ for measuring the upper bound. In line 6, we find a seed of the most correlated keywords to the original query keywords and store them into $\mathcal{K}'$. Then for each $k \in \mathcal{K}'$, we store k into $e.q$ and measure the total score and store it into the $e.\lambda$ in lines 7-9. In lines 10-11, we measure the upper bound $\overline{\lambda}$ for the entry e and push e into the max heap $\mathcal{H}$. In lines 12-13, we measure the lower bound for the current entry e into $\underline{\lambda}$ and compute the approximation rate θ. In line 16, we invoke the procedure *generateQuery* and store the result in e. In lines 18-20, if $\mathcal{Q}^* = l$, we update the top-l list $\mathcal{Q}^*$ and the threshold score λ^{min} respectively. Otherwise, we add the query $e.q$ to $\mathcal{Q}^*$ in line 22.

The procedure *generateQuery* in line 16 of Algorithm 4 is pseudocoded in Algorithm 5. In lines 1-2, if $\mathcal{H} \neq \emptyset$, we pop the top element of $\mathcal{H}$ and store it into e. If $e.\overline{\lambda} < \lambda^{min}$, we stop the search process according to lemma 2 as pseudocoded in lines 3-4. In line 5, if $|e.q| < |q_0|$ we need to add more keywords to $e.q$. In line 6 we read the keywords $k_i \in \mathcal{G}.\mathcal{V}$ that are not a part of $e.q$. Then in lines 7, we add the keyword k_i to $e.q$ and generate the new expanded partial query $e'.q$. In line 8, we compute the total correlation score for the expanded entry and store it into $e'.\lambda$. In line 9, if the score of the expanded entry $e'.\lambda$ is better than the $(1+\theta) \times e.\lambda$ (it means that by adding k_i to $e.q$ the score will increase by $(1+\theta)$ factor), the expanded entry e' is eligible for the search; therefore, we measure the upper bound $e'.\overline{\lambda}$ and push the expanded element into $\mathcal{H}$ for the next iterations

Algorithm 4: Greedy-based Approximation

```
    Input  : $q_0$, $\mathcal{G}$, $\mathcal{S}$, $\alpha$, $l$
    Output: Top-$l$ YMAL Queries $\mathcal{Q}^*$
 1  $\mathcal{Q}^* \leftarrow \emptyset;\mathcal{H} \leftarrow \emptyset$;                                   // initialization
 2  $\mathcal{K} \leftarrow findKeywords(q_0, \mathcal{S})$ ;                         // co-occurring keywords
 3  $\mathcal{G} \leftarrow generateGraph(\mathcal{K})$ ;                                 // keyword graph
 4  $\mathcal{V}' \leftarrow sort(\mathcal{G}.\mathcal{V}, \lambda^{ex})$; $\mathcal{E}' \leftarrow sort(\forall k_i, k_j \in \mathcal{G}.\mathcal{V}, cor(k_i, k_j))$;
 5  $k = findMaxScoreKeyword(\mathcal{G}.\mathcal{V})$; $e.q \leftarrow k$; $e.q \leftarrow k$;
 6  $\mathcal{K}' = findMaxScoreSeeds(\mathcal{G}.\mathcal{V})$;                             // seed keywords
 7  for each $k \in \mathcal{K}'$ do
 8      $e.q \leftarrow k$;      // access the keywords in $\mathcal{K}'$ in ascending order based on $\lambda^{ex}$
 9      $e.\lambda \leftarrow (1-\alpha) \times \lambda^{ex}(k, q_0, T)$;
10      $e.\overline{\lambda} \leftarrow measureUpperBound(e, q_0, \mathcal{G}, \mathcal{V}', \mathcal{E}')$;                  // Eq. 9
11      $\mathcal{H}.push(e)$;                                                  // insert $e$ into $\mathcal{H}$
12  $e.\underline{\lambda} \leftarrow measureLowerBound(e, q_0, \mathcal{G}, \mathcal{V}', \mathcal{E}')$;                      // Eq. 12
13  $\theta \leftarrow \frac{e.\overline{\lambda} - e.\underline{\lambda}}{|\mathcal{G}.\mathcal{V}|}$;                                           // approximation rate
14  $\lambda^{min} \leftarrow 0$; $\mathcal{Q}^* \leftarrow \emptyset$;                                   // initialization
15  while $\mathcal{H} \neq \emptyset$ do
16      $e \leftarrow generateQuery(q_0, \mathcal{H}, \mathcal{G}, \theta, \lambda^{min}, \mathcal{V}', \mathcal{E}')$;
17      if $e \neq null$ then
18          if $|\mathcal{Q}^*| = l$ then
19              $\mathcal{Q}^* \leftarrow update(\mathcal{Q}^*, e.q)$;                      // update top-$l$ with $e.q$
20              $\lambda^{min} \leftarrow minScore(\mathcal{Q}^*)$;                               // update $\lambda^{min}$
21          else
22              $\mathcal{Q}^* \leftarrow \mathcal{Q}^* \cup e.q$;                                   // add query to top-$l$
23      else
24          break;                                    // top-$l$ YMAL queries computed
25  return $\mathcal{Q}^*$
```

in lines 10-11. In lines 14-16, we read the keyword $k_i \in e.q$ and remove it from the query $e.q$ and generate the new shrunk entry e', measure the total score $e'.\lambda$ to check whether the score increases by removing the keyword. In line 17, if the score of the shrunk entry $e'.\lambda$ is better than the $(1+\theta) \times e.\lambda$ (it means that by removing k_i from $e.q$ the score will increase by $(1+\theta)$ factor), the shrunk entry e' is eligible for the search; therefore, we measure the upper bound $e'.\overline{\lambda}$ and push the shrunk element into $\mathcal{H}$ for the next iterations in lines 18-19. In line 21, we return e for the procedure *generateQuery*.

5 Discussion

Query Size. When generating a YMAL query q, adding or removing a keyword may increase its score $\lambda(q, T)$. Therefore, the size of the YMAL query q may affect its score. However, to check all the sizes for a YMAL query, we have to generate all the subsets from the keyword pool $\mathcal{K}$ which is impractical. Moreover, when the size of the query is extremely small or large, it generates very loose or tight results which is not appropriate. From our observation, when we set the YMAL query size to $|q_0|$, we get most of the top-l YMAL queries. It also reflects the user original search intentions for the query size. However, it is possible to generate some of the top-l YMAL queries by increasing or decreasing the size of q (from $|q_0|$) by a small number of keywords. One way to generate the top-l YMAL queries of any

Algorithm 5: generateProcedure for Greedy-based Approximation

```
Input  : $q_0$, $\mathcal{H}$, $\mathcal{G}$, $\theta$, $\lambda^{min}$, $\mathcal{V}'$, $\mathcal{E}'$
Output: $e$ element containing YMAL Query
1  while $\mathcal{H} \neq \emptyset$ do
2      $e \leftarrow \mathcal{H}.pop()$;
3      if $e.\overline{\lambda} < \lambda^{min}$ then
4          return $null$;                                      // as per Lemma 2
5      if $|e.q| < |q_0|$ then
6          while $k_i = getNext(\mathcal{G}.\mathcal{V}) \neq \emptyset$ and $k_i \cap e.q = \emptyset$ do
7              $e'.q \leftarrow e.q \cup k_i$;                  // add keyword to the query
8              $e'.\lambda \leftarrow \alpha \times \lambda^{in}(e'.q, T) + (1-\alpha) \times \lambda^{ex}(e'.q, q_0, T)$;
9              if $e'.\lambda > (1+\theta) \times e.\lambda$ then
10                 $e'.\overline{\lambda} \leftarrow measureUpperBound(e', q_0, \mathcal{G}, \mathcal{V}', \mathcal{E}')$;
11                 $\mathcal{H}.push(e')$;
12     else
13         $flag \leftarrow true$;                               // boolean flag
14         while $k_i = getNext(e.q) \neq \emptyset$ do
15             $e'.q \leftarrow e.q \setminus k_i$;              // exclude keyword from $e.q$
16             $e'.\lambda \leftarrow \alpha \times \lambda^{in}(e'.q, T) + (1-\alpha) \times \lambda^{ex}(e'.q, q_0, T)$;
17             if $e'.\lambda > (1+\theta) \times e.\lambda$ then
18                 $e'.\overline{\lambda} \leftarrow measureUpperBound(e', q_0, \mathcal{G}, \mathcal{V}', \mathcal{E}')$;
19                 $\mathcal{H}.push(e')$; $flag \leftarrow false$;
20         if $flag$ then
21             return $e$;
```

size is to start from the top-l YMAL queries of size $|q_0|$. From the output of either of Algorithm 1 (for A* search-based technique) or Algorithm 4 (for greedy-based approximation), we can select the YMAL query q with the highest score each time, then add or remove a keyword to see if the score is increased (for Algorithm 1) or increased by the proportion of the approximation rate (for Algorithm 4). Another way to generate the top-l YMAL queries of any size is to update Algorithm 1 or Algorithm 4 by removing the cardinality constraint (i.e., fix the size to $|q_0|$). In other words, for Algorithm 1, a YMAL query candidate q is generated when adding a keyword would not increase the score $\lambda(q, T)$. For Algorithm 4, a YMAL query candidate q is generated when adding or removing a keyword would not be increased the score by $\theta \times \lambda(q, T)$.

Effect of Parameters γ and η. γ parameter tunes the correlation value of the keywords between external and internal sources. If we prefer to use a local data source correlation estimation, we can set γ to a lower value. Otherwise, we can set γ to bigger value to reflect the correlation globally. Also η setting can affect the performance. If we set η to smaller value, the co-occurring graph would be more connected and this can negatively affect the performance. However, when we set η to a bigger value, it can improve the performance but it will affect the effectiveness negatively. In this paper, we find $\eta \geq 0.2$ is fairly good. The effect of α is discussed in the experiments. Note that our proposed framework for presenting YMAL queries is not XML exclusive. Hence, the proposed framework for presenting top-l YMAL queries can be easily adapted to work with other types of data, such as RDF and so on.

Table 2: Sample queries and their top-1 YMAL queries

Dataset	q_0	$Top - 1\ YMAL\ query$
IMDB	alpacino,coppola,drama	caan,coppola,drama
IMDB	drama,2012,crowe	action,2012,crowe
IMDB	gladiator,action,english,combat	braveheart,action,english,revolt
IMDB	shrek,animation,myers	madagascar,animation,miller
IMDB	scream,horror,telephone,killer	ring,horror,ghost,killer
DBLP	stonebraker,sigmod,2017	stonebraker,pvldb,2017
DBLP	web, search, semantic	web, search, text
DBLP	www, conference, 2016,multimedia	www, conference, 2016,exploration
DBLP	agrawal,keyword,search	matteis,keyword,search
DBLP	www,search,entity,2016	www,search,experience,2016

6 Experiments

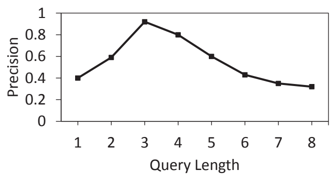
In this section, we evaluate the effectiveness and efficiency of our proposed methods for presenting top-l YMAL queries to users. The proposed methods are as follows: (a) NAIVE method (the brute force method), (b) ASTAR method, and (c) GREEDY method. The experiments are conducted on two real datasets: (a) Internet Movie Database (IMDB) 170 MB, and (b) Digital Bibliography and Library Project (DBLP) 700 MB. All of our algorithms are implemented in *C#* and the experiments are carried out using a PC with 3.2 GHz CPU, 8 GB memory and running on 64-bit windows 7. We conduct our experiments on a wide range of tested queries. A part of the tested queries is presented in Table 2.

6.1 Effectiveness

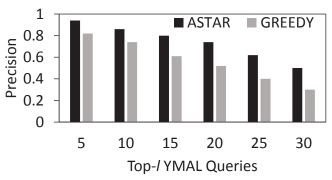
This section evaluates the effectiveness of the proposed methods.

6.1.1 User Study

We conducted a comprehensive user study to evaluate the quality of our YMAL queries. We selected our users among experts and ordinary users and ask them to score each of our top-l YMAL queries between [0-1] based on their relevance to the original query (0 means the least relevance and 1 means the most relevance). Fig. 5(a) presents the precision of YMAL queries based on their size. Clearly when the size of the queries is small ($|q| = 1$ or $|q| = 2$), the precision is low. When the queries size increases to 3 and 4, the precision increases. Finally, when the queries size is bigger than 4, the precision deteriorates. That's probably for the fact that when we generate queries with small size, the query may not reflect all the original query properties. Conversely, when we generate a query with a big size, the query may contain some keywords that are not relevant enough to the original query. Fig. 5(b) presents the precision of YMAL queries for ASTAR and GREEDY methods. Clearly, when the top-l size increases, the precision of the queries decreases for both methods which shows that our ranking scheme successfully score the more relevant YMAL queries with higher scores. The precision in ASTAR is better than the GREEDY because the ASTAR generates the optimal YMAL queries while the GREEDY method generates the approximate suboptimal queries. However, the precision of the GREEDY method is not deteriorated with big margin.



(a) Query size effect



(b) Precision comparison

Fig. 5: Quality of YMAL queries

Table 3: Trading off between correlation scores

Dataset	*Method*	α	$AVG(\lambda^{in})$	$AVG(\lambda^{ex})$	$AVG(\lambda)$
IMDB	ASTAR	0.2	2.819	1.587	1.834
IMDB	ASTAR	0.8	2.858	1.546	2.596
IMDB	GREEDY	0.2	2.747	1.588	1.82
IMDB	GREEDY	0.8	2.753	1.584	2.315
DBLP	ASTAR	0.2	2.479	1.563	1.746
DBLP	ASTAR	0.8	2.784	1.402	2.507
DBLP	GREEDY	0.2	2.423	1.563	1.737
DBLP	GREEDY	0.8	2.439	1.559	2.262

6.1.2 Effect of tuning parameter α

We provide a case study to show the effect of the tuning parameter α on the YMAL query score and how this parameter is a trade-off between the internal and external correlation scores. Table 3 presents the average internal, external and total correlation scores for the top-10 YMAL queries retrieved from the results of the tested queries in IMDB and DBLP. Clearly, the external correlation score is large in both ASTAR and GREEDY when $\alpha = 0.2$ for both datasets. That's because when we set α to a small number, more emphasis is put on the external correlation and consequently, more similar keywords to the original query are selected to generate the YMAL queries. However, when we set α to a big number, we put more priority on the internal correlation score; therefore, the YMAL queries are generated by using the keywords with the better internal correlation scores.

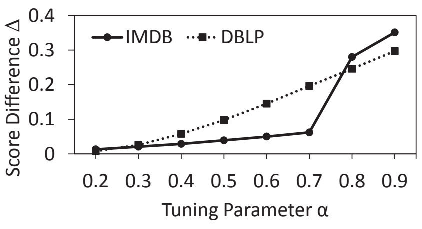
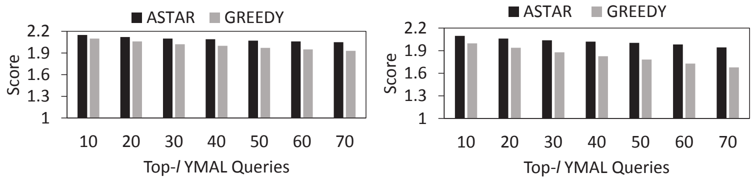


Fig. 6: Score differences between ASTAR and GREEDY

Fig. 6 presents the average score difference Δ of GREEDY method compared with ASTAR method for computing top-10 YMAL queries retrieved for the tested queries in IMDB and DBLP datasets. From the figure, we observe that the Δ grows when we set $\alpha = 0.2$ to 0.9 for both methods on both datasets. That's because the GREEDY method starts from the most correlated keyword to the original keywords; therefore, when we set α to larger numbers the GREEDY method score is more affected by its low internal correlation. In IMDB dataset, from $\alpha = 0.7$ to 0.8, we observe a big jump in Δ because at this point the GREEDY method fails to find a suboptimal YMAL query that its internal correlation score is close enough to the YMAL query which is generated by ASTAR. Therefore, the GREEDY method score is heavily affected by this difference. However, when the α is balanced (for instance $\alpha = 0.5$), the performance of GREEDY method is close to that of ASTAR.



(a) IMDB (b) DBLP

Fig. 7: Effect of top-l size on the effectiveness

6.1.3 Effect of $|l|$

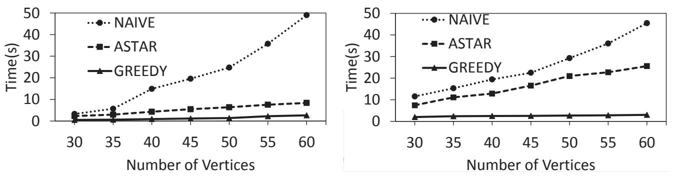
Fig. 7 presents the effect of $|l|$ on the proposed methods effectiveness for IMDB and DBLP. From the figure, we observe that the average total score for the top YMAL queries generated by the GREEDY method is very close to the ASTAR method for $l \leq 20$ for both datasets. For $l > 20$, the score difference Δ between the GREEDY and ASTAR methods grows bigger. That's because the GREEDY method finds the suboptimal YMAL queries that their score is locally maximum and when the parameter l increases, the average score of these locally maximum YMAL queries may get smaller than the ASTAR YMAL queries. However, for the smaller values of l, GREEDY is not much affected.

6.2 Efficiency

In this section, we evaluate the efficiency of our proposed methods from various perspectives.

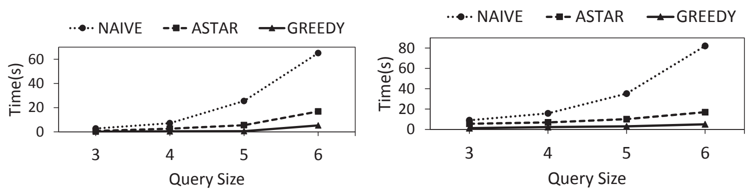
6.2.1 Effect of $|\mathcal{K}|$

As our keyword pool size $|\mathcal{K}|$ is crucial for the performance of the proposed methods, we conduct experiments to present the effect of the keyword pool size on the efficiency of our proposed methods. Fig. 8 presents the execution time for different methods by varying the keyword pool size $|\mathcal{K}|$ on IMDB and DBLP datasets. From Fig. 8(a), we observe that the NAIVE method is more sensitive to $|\mathcal{K}|$ since it generates all possible queries from $\mathcal{K}$; however, ASTAR and GREEDY methods are not that sensitive to $|\mathcal{K}|$. In Fig. 8(b), the execution time for the NAIVE and ASTAR methods show a jump when $30 \leq |\mathcal{K}| \leq 50$. For $|\mathcal{K}| \geq 55$, the execution time for ASTAR increases with a lower rate. That's because the ASTAR method estimation overhead affects its performance at first (specially when the estimation is loose) and then it becomes better in performance because the estimation



(a) IMDB (b) DBLP

Fig. 8: Effect of $|\mathcal{K}|$ on efficiency



(a) IMDB (b) DBLP

Fig. 9: Effect of query size on the efficiency

becomes tighter which improves its performance. In NAIVE method; however, it finds all the possible queries from $\mathcal{K}$ to generate the YMAL queries $\mathcal{Q}$ which negatively affects its performance. Therefore, we conclude that NAIVE method is not scalable while GREEDY method has the best scalability with respect to $|\mathcal{K}|$.

6.2.2 Effect of query size

In this experiment, we analyze the effect of query size on the performance of the proposed methods. To conduct the experiment, we set the original tested queries size to $3, 4, 5$, and 6 and we measure the average execution time for generating top-5 YMAL queries. Fig. 9 presents the corresponding results for IMDB and DBLP. From the figure, NAIVE method execution time exhibits a sharp increase when the query size increases on both datasets. That's because the NAIVE method builds all the combination of $\mathcal{K}$ with size $|q_0|$ to generate the YMAL queries; therefore, by increasing the query size we observe an explosion in the execution time. Similarly in the ASTAR and GREEDY methods, the execution time grows when the query size increases; however, the growth is slower than the NAIVE method. We conclude that NAIVE method is highly sensitive to this parameter while GREEDY method is the least sensitive method w.r.t. to query size.

7 Related Work

A recommendation system usually issues exploratory queries on the data source to generate recommended items to the user [8,2]. There are two major types of recommendation methods in the literature: (a) recommendation by exploring similar items to those that are visited before [16,19]; and (b) recommendation by exploring the items that are relevant to those that are liked by the user in the past [1,18]. In [10], a framework that supports the query-from-examples is presented for exploration over large datasets. This framework employs active learning to construct exploratory queries based on user's preferences. Some works propose a multi-objective view recommendation scheme for visual data exploration [9]. Most of these works focus on relational databases. However, XPloreRank works on semi-structured data and constructs YMAL queries based on user original query result content.

The query refinement focuses on modifying the original query to assist the user to find her desired results. Sometimes the original query retrieves many results that are less relevant to the user query intentions. Therefore, query refinement methods add some constraints to the original query to limit the number of results [20]. Conversely, when the user query retrieves no result or insufficient results, the query refinement methods relax the original query constraints to generate more results for the user [17, 3, 12]. Some research works propose the query refinement techniques to discover and solve the mismatch problem when the original query retrieves erroneous results [5]. However, XPloreRank analyzes the user original query results content to generate highly correlated YMAL queries for recommendation.

8 Conclusion

In this paper, we investigate exploratory search on XML keyword search. Our approach exploits the co-occurring content within the user original query results to generate YMAL queries. The YMAL queries produce recommended additional results that may interest the user. These queries are ranked based on their capability to produce meaningful results with respect to the data source and their similarity to the user original keyword query. Since the number of YMAL queries is enormous, we propose an A* search-based technique to avoid generating all the YMAL queries and conduct a search on the co-occurring keywords space. Also, we propose a greedy-based approximation for the A* search-based technique to improve the performance of the A* search substantially and return the top-l YMAL queries to the user. The extensive experiments verify the effectiveness and efficiency of our approach.

References

1. G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.
2. J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman. SQL querie recommendations. *PVLDB*, 3(2):1597–1600, 2010.
3. A. Baid, W. Wu, C. Sun, A. Doan, and J. F. Naughton. On debugging non-answers in keyword search systems. In *EDBT*, pages 37–48, 2015.
4. Z. Bao, T. W. Ling, B. Chen, and J. Lu. Effective xml keyword search with relevance oriented ranking. In *ICDE*, pages 517–528, 2009.
5. Z. Bao, Y. Zeng, T. W. Ling, D. Zhang, G. Li, and H. V. Jagadish. A general framework to resolve the mismatch problem in XML keyword search. *VLDB J.*, 24(4):493–518, 2015.
6. R. Cilibrasi and P. M. B. Vitányi. The google similarity distance. *IEEE Trans. Knowl. Data Eng.*, 19(3):370–383, 2007.
7. S. Cohen and T. Brodianskiy. Correcting queries for XML. *Inf. Syst.*, 34(8):690–710, 2009.
8. M. Drosou and E. Pitoura. Ymaldb: exploring relational databases via result-driven recommendations. *VLDB J.*, 22(6):849–874, 2013.
9. H. Ehsan, M. A. Sharaf, and P. K. Chrysanthis. Muve: Efficient multi-objective view recommendation for visual data exploration. In *ICDE*, pages 731–742, 2016.

10. X. Ge, Y. Xue, Z. Luo, M. A. Sharaf, and P. K. Chrysanthis. REQUEST: A scalable framework for interactive construction of exploratory queries. In *IEEE BigData*, pages 646–655, 2016.
11. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD*, pages 16–27, 2003.
12. M. S. Islam, C. Liu, and R. Zhou. Flexiq: A flexible interactive querying framework by exploiting the skyline operator. *Journal of Systems and Software*, 97:97–117, 2014.
13. H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD*, pages 13–24, 2007.
14. F. Li and H. V. Jagadish. Usability, databases, and HCI. *IEEE Data Eng. Bull.*, 35(3):37–45, 2012.
15. C. Mishra and N. Koudas. Interactive query refinement. In *EDBT*, pages 862–873, 2009.
16. R. J. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *ACM Conference on Digital Libraries*, pages 195–204, 2000.
17. U. Nambiar and S. Kambhampati. Answering imprecise queries over autonomous web databases. In *ICDE*, 2006.
18. C. Palmisano, A. Tuzhilin, and M. Gorgoglione. Using context to improve predictive modeling of customers in personalization applications. *IEEE Trans. Knowl. Data Eng.*, 20(11):1535–1549, 2008.
19. M. J. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.
20. R. Schenkel, A. Theobald, and G. Weikum. Semantic similarity search on semistructured data with the XXL search engine. *Inf. Retr.*, 8(4):521–545, 2005.
21. C. Sun, C.-Y. Chan, and A. K. Goenka. Multiway slca-based keyword search in xml data. In *World Wide Web*, pages 1043–1052, 2007.
22. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest lcas in xml databases. In *SIGMOD*, pages 527–538, 2005.
23. Y. Xu and Y. Papakonstantinou. Efficient lca based keyword search in xml data. In *EDBT*, pages 535–546, 2008.
24. R. Zhou, C. Liu, and J. Li. Fast elca computation for keyword queries on xml data. In *EDBT*, pages 549–560, 2010.