Data & Knowledge Engineering 113 (2018) 1–17

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Thematic ranking of object summaries for keyword search

Georgios J. Fakas[a], Yilun Cai[b], Zhi Cai[c,*], Nikos Mamoulis[d]

[a] Department of Information Technology, Uppsala University, Sweden
[b] Lenovo Group Limited, Hong Kong
[c] Beijing Advanced Innovation Center for Future Internet Technology, Faculty of Information Technology, Beijing University of Technology, China
[d] Department of Computer Science and Engineering, University of Ioannina, Greece

ARTICLE INFO

Keywords:
Keyword search
Object summaries
Top-*k* Queries
Relational databases

ABSTRACT

An Object Summary (OS) is a tree structure of tuples that summarizes the context of a particular Data Subject (DS) tuple. The OS has been used as a model of keyword search in relational databases; where given a set of keywords, the objective is to identify the DSs tuples relevant to the keywords and their corresponding OSs. However, a query result may return a large amount of OSs, which brings in the issue of effectively and efficiently ranking them in order to present only the most important ones to the user.

In this paper, we propose a model that ranks OSs containing a set of identifying keywords (e.g., `Chen`) according to their relevance to a set of thematic keywords (e.g. `Mining`). We argue that the effective thematic ranking of OSs should combine gracefully IR-style properties, authoritative ranking and affinity. Our ranking problem is modeled and solved as a top-*k* group-by join; we propose an algorithm that computes the join efficiently, taking advantage of appropriate count statistics and compare it with baseline approaches. An experimental evaluation on the DBLP and TPC-H databases verifies the effectiveness and efficiency of our proposal.

1. Introduction

The keyword search paradigm in relational databases (R-KwS) [1,2] extracts trees of tuples that collectively contain a set of keywords and they are connected through foreign-key links. For example, the query {"Chen", "Agrawal"} on the DBLP database will return the papers co-authored by authors *Chen* and *Agrawal*. On the other hand, the R-KwS paradigm may not be very effective when trying to extract information about a particular data subject (DS),[1] e.g. for author *Chen*, as the result will include only single tuples containing the keyword, (i.e., author tuples of *Chens*). To address this issue, in [3–5], the concept of Object Summary (OS) is introduced. An OS summarizes all data held in a database about a particular DS. More precisely, an OS is a tree with the tuple t^{DS} containing the keyword(s) (e.g. Author tuple *Ming-Syan Chen*) as the root node and its neighboring tuples, containing additional information (e.g. his papers, co-authors etc.), as child and descendant nodes. Fig. 1 illustrates part of the OS for *Ming-Syan Chen*.

OSs can be very useful for users searching for comprehensive information about DSs related to a set of keywords. On the other hand, a query may be related to numerous DSs; thus computing and showing all the corresponding OSs could overwhelm the user. For example, the query "Chen" on the DBLP database would return around 1,982 OSs, as there are that many authors having "Chen"

* Corresponding author.
E-mail addresses: georgios.fakas@it.uu.se (G.J. Fakas), ycai3@lenovo.com (Y. Cai), caiz@bjut.edu.cn (Z. Cai), nikos@cs.hku.hk (N. Mamoulis).

[1] A DS represents an individual who is a subject of personal data. In this work, a DS has a broader meaning and may represent any object (e.g. a product, order, etc.) that is a subject of data.

https://doi.org/10.1016/j.datak.2017.08.002
Received 18 September 2016; Received in revised form 24 June 2017; Accepted 18 August 2017
Available online 28 October 2017
0169-023X/

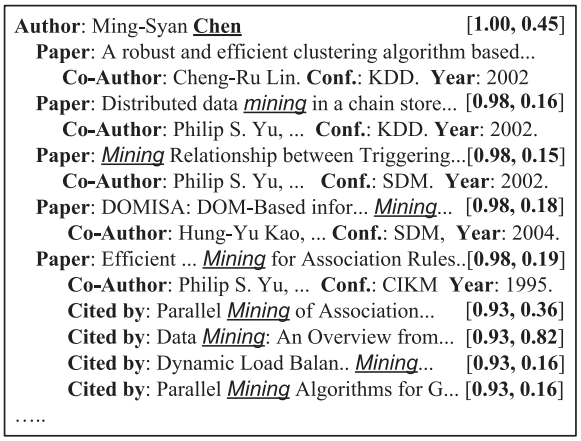

Fig. 1. A fraction of the *Ming-Syan Chen* OS ($dl(OS) = 6,376$, $tf_{Mining}(OS) = 294$, $[Af(t), Im(t)]$).

as a component of their names. These OSs can be ranked according to their importance, which is calculated by aggregating scores of the OS contents (i.e., tuples) based on *authoritative ranking* and *affinity* [5]. Authoritative ranking facilitates the ranking of tuples by considering the flow of authority via their semantic connections. Affinity measures the closeness of two tuples by considering their distance, connectivity, etc. However, such a ranking is very static; e.g., *Peter Chen* will always be ranked first because of his many citations. This is ineffective for users who search for a DS that does not have the best importance scores. In view of this, in this paper, we propose the *thematic ranking* of OSs, where thematic keywords are also input by the user. Specifically, the user inputs (1) a set of identifying keywords (e.g. {"Chen"}) and (2) a set of thematic keywords (e.g. {"mining"}). In our example, while *Ming-Syan Chen* fails to compete *Peter Chen* when only using the identifying keyword "Chen", the use of the additional thematic keyword "mining" makes *Ming-Syan Chen* prevail, because his OS contains many times the thematic word.

There is already a plethora of relevant research in R-KwS ranking. Existing ranking approaches primarily consider the size (in tuples) of the results as well as the IR-style metrics, e.g. [6]. However, the straightforward adaptation of these paradigms (as we explain in detail in Section 2) is inappropriate for the effective ranking of OSs. For instance, a R-KwS result with a small size in tuples has a higher ranking score; in contrast, an author OS with many papers should have a higher importance than another author with very few papers. Also, existing work in R-KwS ranking is limited to the consideration of IR approaches which disregards authoritative flow through relationships and affinity. We argue that the importance and affinity of tuples should be combined with IR techniques.

In this paper, we consider an OS as a virtual document. Given an identifying and a thematic keyword, we propose that the thematic ranking of an OS should gracefully combine (1) the *importance of the data subject* of the OS, (2) *IR properties*, *importance* and *affinity* of the thematic tuples in the OS (i.e. the tuples containing the thematic keywords). Namely, the high importance of the DS in combination with the frequent occurrences of thematic tuples (which have high importance and affinity) in an OS should result to a high thematic score. Note that IR properties also consider (among other metrics) the amount of words per OS and the respective average of all OSs. Considering the query ⟨{"Chen"}, {"Mining"}⟩ (Fig. 1), an important *Chen* OS with many citations who has authored many well cited papers including the thematic word should get a high score. Comparatively, the thematic keyword found in non-cited papers will result to a smaller thematic score. Also, affinity influences accordingly the thematic score; for instance, consider the affinity of a keyword which is found in the title of a paper authored by *Chen* in comparison to a keyword found in a paper that *Chen* cites in one of his papers.

Motivation: OSs can have many applications. For instance, the OS results are in more analogy than the R-KwS paradigms' results to the web keyword search results (e.g. Google). For instance, the example of Fig. 1 resembles a web page (as both include comprehensive information about the DS). Therefore, for non-technical users with experience only on web search engines, the OS paradigm will be closer to their expectations. In general, an OS is a concise summary of the *context* around any pivot database tuple or graph node, finding application in (interactive) data exploration, schema extraction, etc. Furthermore, the thematic ranking (regardless of OSs) finds additional applications; given any graph (e.g., bibliographic, social network, semantic knowledge [7,8], linked data [9], etc.), we can rank nodes according to their importance in combination with the relevance of the data around them to thematic keywords. For example, consider a data-graph with authors; we can rank authors based on the theme of their papers or books (e.g. an author who has written all his 10 books in databases is thematically more important than another author authored 9 books in AI and 1 in databases).

Challenges and approach: The combined formula that we propose is non-monotonic with respect to the scores of the constituent OSs tuples. In addition, it dictates the implementation of an expensive top-k Group By join (denoted as kGBJ): the join paths between each DS tuple and all the thematic tuples that link to it have to be *grouped* before the score of the DS (and the corresponding OS) can be computed. As a first attempt to tackle our problem, we propose to pre-compute wherever possible the proposed formula's parameters. During online processing, we need to aggregate the joins of the thematic tuples (i.e. the tuples containing the thematic keywords q_2, denoted as $R_j^{\mathrm{Th}}(q_2)$) per DS, i.e. for each tuple containing the identifying keywords q_1, denoted as

$R_i^{DS}(q_1)$. We also denote as R_j^{Th} and R_i^{DS} the respective relations that $R_j^{Th}(q_2)$ and $R_i^{DS}(q_i)$ belong to; for simplicity, we drop the subscripts when the context is clear, i.e. R^{Th}, R^{DS}, $R^{Th}(q_2)$ and $R^{DS}(q_i)$ respectively. Namely, the kGBJ of $R_i^{DS}(q_1)$ with $R_j^{Th}(q_2)$ operator. As we explain in more detail in Section 2.2, the kGBJ problem differentiates technically from existing top-k R-KwS ranking algorithms which consider top-k ranking of joins of tuples containing the keywords. The efficient calculation of kGBJ can be very challenging as $R_i^{DS}(q_1)$, $R_j^{Th}(q_2)$, $|OS|$ (the size of an OS in tuples) sizes can be very large and also the join path can also be long (in comparison to similar problems studied in earlier work e.g. [6,10,11]). For instance, R^{DS}_{Author}(Chen) = 1, 982, R^{Th}_{Paper}(mining) = 2, 961, *Ming-Syan Chen* $|OS|$ = 941 and a join path can be as long as Author → Paper ← ConfYear ← Year. Hence, a typical query may require the join of tens of thousands of tuples. We investigate for the first time these challenging problems in the context of relational keyword search of large initial inputs (especially on long join paths).

Contributions: In summary the contributions of this paper are as follows:

- The effective thematic ranking of OSs. A formula that gracefully combines IR, affinity and authoritative importance of tuples containing themes is proposed.
- Baseline and optimized algorithms that address the top-k thematic ranking of OSs. Our formula requires a kGBJ operator, the efficient evaluation of which has not been studied before, to the best of our knowledge. We propose an efficient top-k bidirectional algorithm for this problem, which can also be used for answering general kGBJ queries in relational databases.
- An experimental evaluation which verifies the effectiveness of the proposed ranking paradigm and the efficiency of the proposed algorithm.

Roadmap: The rest of the paper is structured as follows. Section 2 describes background and related work. Section 3 introduces the semantics of the thematic ranking and the respective formulas. Section 4 presents preprocessing techniques, whereas Section 5 proposes methods for thematic ranking. Section 6 studies the thematic ranking of OSs, in the case where the thematic keywords are included in multiple relations. Section 7 presents a thorough investigation of the effectiveness of thematic OS ranking and the efficiency of the proposed algorithms. Finally, Section 8 provides concluding remarks.

2. Background and related work

In this section, we first describe the concept of object summaries (OSs), which we build upon in this paper. We then present and compare related work in R-KwS and top-k ranking. To the best of our knowledge there is no previous work that focuses on the thematic ranking of OSs.

2.1. Object summaries

In the context of OS search in relational databases [5], a query is a set of keywords (e.g. {"Chen"}) and the result is a set of OSs. An OS is generated for each tuple (t^{DS}) found in the database that contains the keyword(s) as part of an attribute's value (e.g. tuple *Chen* of relation Author in the DBLP database). t^{DS} is called the *data subject* (DS) tuple and stores the main semantics of the OS. An OS is a tree structure composed of tuples, having t^{DS} as root and t^{DS}'s neighboring tuples (i.e., those associated through cascading foreign key links) as its children/descendants. The rationale is that there are relations, denoted as R^{DS} (where $t^{DS} \in R^{DS}$; e.g. the Author relation), which hold information about the queried DSs and the relations linked around R^{DS} contain additional information about each of the DSs. For each R^{DS}, a Data Subject Schema Graph (G^{DS}) can be generated; this is a directed labeled tree that captures a subset of the database schema with R^{DS} as a root. For example, given the DBLP schema as shown in Fig. 3, Fig. 2 illustrates the G^{DS} of relation Author. In simple words, G^{DS} is a "treelization" of the schema, where R^{DS} becomes the root, its neighboring relations become child nodes and relations involved in loops or many-to-many relationships are replicated. For example, relations PaperCitedBy, PaperCites and Co-Author on Author G^{DS} are replicas of Paper and Author relations. In order to

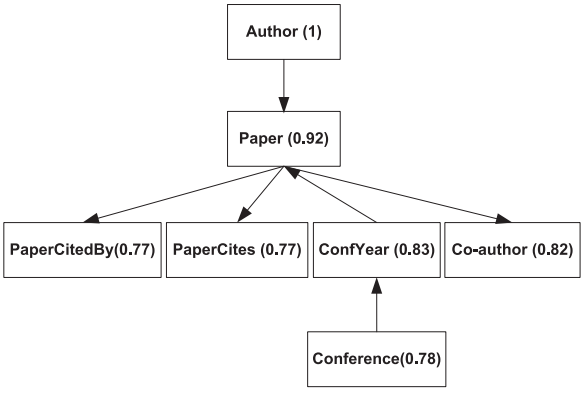

Fig. 2. DBLP Author G^{DS} (Affinity).

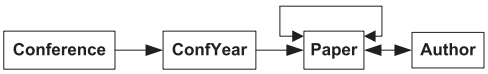

Fig. 3. The DBLP database schema.

constrain the size of G^{DS} (which could be infinite due to the existence of loops), an *affinity* measure $Af(R_i)$ on each relation R_i to R^{DS} is defined and used. In simple words, the affinity of R_i to R^{DS} is large if there are few and short paths from R_i to R^{DS} in G^{DS} (see [5] for a precise definition). Finally, G^{DS} is restricted to contain only R_i's whose affinity to R^{DS} exceeds a certain threshold θ. By traversing the resulting G^{DS} (i.e., by joining the relations in the paths of G^{DS}), we can generate the OSs. For instance, for the keyword query "Chen", the Author G^{DS} of Fig. 2, and $\theta = 0.7$ the OS presented in Fig. 1 will be generated.

In [5] the static ranking of OSs was investigated by considering only the identifying keywords. More precisely, the proposed ranking formula considers the *importance* of the constituent tuples and OS size. As already mentioned, such a ranking is not effective in finding a specific OS with a relatively low importance of a specific theme. Finally, the retrieval of concise and informative OSs (denoted as size-l OSs) was investigated in [12] while the diverse and proportional size-l OS retrieval was investigated in [13,14]. In these works, the local importance score of tuples (Eq. (5)) was used in order to decide which tuples are the most important to include in size-l OSs. Although, in this work we reuse this ranking measure for tuples, the problem definitions are orthogonal directions to the objective of our present work.

2.2. Keyword search in databases (R-KwS)

R-KwS techniques [1,2] discover a sets of tuples, which are collectively relevant to a set of keywords and they are connected via foreign key links. For instance, the R-KwS query {"Chen", "VLDB"} finds all papers authored by *Chen* that are published at *VLDB*, or cited or citing papers at *VLDB* etc. For this purpose the concept of *candidate network* (CN) is introduced to model a part of the schema graph connecting relations containing the keywords. For example, if R^{Paper}(Chen) and $R^{Conference}$(VLDB) are tuple sets in relations R^{Paper} and $R^{Conference}$ containing keywords "Chen" and "VLDB", respectively, then a CN could be the $CN_1 = R^{Author}(\text{Chen}) - R^{Paper} - R^{ConfYear} - R^{Conference}(\text{VLDB})$. Evidently, the R-KwS paradigm differs from OS search semantically, since it focuses on the ranking of CNs that connect the given keywords, whereas OSs are trees centered around the data subject described by the keywords. R-KwS techniques also have technical differences with our present problem (i.e., thematic ranking of OSs), as we discuss in the next paragraph.

Technical differences with R-KwS top-k ranking: Top-k R-KwS (e.g. [1,2]) considers mainly IR-style techniques to rank R-KwS results, e.g., TF-IDF scores of the contained keywords in tuples (we shall discuss such IR scoring parameters in our ranking model in detail in Section 3). The objective is to avoid computing CNs at their entirety; the computation of networks of tuples that have low scores (calculated as a function of the IR scores of the tuples that contain the keywords and the distance between them) is avoided as much as possible. The proposed top-k algorithms process the tuples that contain the keywords in descending order of their initial atomic IR scores; they start producing results and concurrently calculate an upper bound score for the tuple networks not computed yet, until the top-k results can be safely reported. For instance consider CN_1; we need to examine whether tuples from R^{Paper}(Chen) and $R^{Conference}$(VLDB) join together and at the same time avoid join results of collectively low IR score. The main difference between our work and these algorithms is that, in order to rank the OSs, we first have to group all join results that have the same data subject together and then rank the groups (thus, a *top-k Group By rank join* denoted as kGBJ). In other words, all join paths that contain a given data subject tuple $t^{DS} \in R^{DS}$ on the one end and a thematic keyword on the other are grouped together and their scores are aggregated before we can rank the t^{DS}'s to derive the final result. Moreover, we consider additional factors in the ranking, such as affinity, the size of the OS corresponding to t^{DS}, etc.

Other related work in R-KwS: Early work in keyword search [15] considers proximity-based ranking, in a similar manner as our thematic ranking of OSs. Proximity-based search is defined by two sets of keywords: A *Find* query specifies a set of objects (Find) that are of potential interest and a *Near* query specifies a Near set. The objective is to rank objects in Find according to their distance to objects in Near. For instance, query "find papers near {"Chen", "Agrawal"}" will return tuples in Paper ranked based on their proximity to keywords "Chen" and "Agrawal". Thus, a co-authored paper which has distance one from both keywords in Near will be ranked higher than a paper not written by *Chen* and/or *Agrawal* but cited by some of their papers. Evidently, this search paradigm differs semantically and technically from our work, because ranking is not based on aggregation and/or IR scores (and also their results are different).

Précis queries [16,17] resemble size-l OSs [5,12] as they append additional information to the result of a R-KwS (i.e., the nodes containing the keywords), by considering neighboring relations that are implicitly related to the keywords. Since R-KwS itself is unrelated to thematic ranking of OSs, Précis queries are also not related to our present work. Other related work includes top-k-size keyword search on tree structured data, e.g. [18,19], combination of user and databases perspectives [20], etc.

2.3. Top-k ranking

Early work on top-k ranking [21–23] addressed the merging of ordered lists using monotonic aggregate functions. All these approaches focus on ranking of objects based on their various attribute values. The more general problem of ranking the results of a join was studied in [24]. Our problem involves grouping and aggregating join results before ranking them. Related work on ranking results of group-by queries is studied in [25], where a group is formed for every distinct pair of tuples from two relations. For

instance, consider two relations R_1 and R_2 containing m and n distinct tuples and assume that each tuple in R_1 can join with any tuple in R_2; then $m \times n$ groups will be generated. Our problem is different, because we group the scores of thematic tuples w.r.t. tuples in the data subject relation only. In general, previous work assumes that (i) both groups and their aggregated data reside in the same relation and/or (ii) the data to be aggregated are directly associated with their groups (i.e., they include the group ID as an attribute) and thus can easily be grouped. Because of these assumptions, it is very easy to calculate the cardinality of each group and use it to accelerate search, while in our case we may have to evaluate expensive joins for this purpose.

3. Thematic ranking of OSs

In this section, we introduce our proposed thematic OS ranking paradigm. A query Q comprises two sets of keywords, namely $Q = \langle q_1, q_2 \rangle$, where (i) q_1 is a set of *identifying* keywords used to generate OSs as defined in [4] and (ii) q_2 is a set of *thematic* keywords facilitating a biased ranking of the OSs of data subjects related to q_1. For instance, for $Q = \langle \{\text{“Chen”}\}, \{\text{“mining”}\} \rangle$, the result in the DBLP database will be a set of OSs, each centered at a *Chen* t^{DS} (i.e., data subject tuple), ranked according to the importance of the t^{DS} (i.e., authors whose name contains “Chen”) and the occurrences of “mining” in the OS. This way, the OS of a less important DS could be ranked high if it is very relevant to the thematic keywords.

For the thematic ranking of OSs, we treat each OS as a *virtual document* with the respective relational semantics. We observe that the following criteria can be used:

(1) The **global Importance** of constituent tuples and especially t^{DS}. We measure global Importance using authority flow techniques. More precisely, we use ObjectRank [26] and ValueRank [4] (other methods can also be investigated). For instance, ObjectRank [26] is an extension of PageRank on databases and introduces the concept of Authority Transfer Rates between the tuples of each relation. This is based on the observation that solely mapping a relational database to a graph (as in the case of the web PageRank) is not accurate and Authority Transfer Rates are required to control the flow of authority in neighboring tuples. Namely, using transfer rates we can favour accordingly authority transfer. For instance, an author will get higher ranking because his papers are well cited by other papers and not because his papers cite other papers.

For example, *Peter Chen* is more important than *David Chen* because he has received more citations; thus, the *Peter Chen* t^{DS} should be given higher ranking. Similarly, if the tuples of the OS containing the thematic keywords have higher global importance (e.g., an important paper on data mining with many citations), the OS should be given higher score.

(2) The **IR-properties** of the thematic keywords in OSs. The frequency of the appearance of the thematic keywords in the tuples of the OSs can be very useful in the biased ranking of OSs. Using TF-IDF semantics, these frequencies should be normalized to favor cases of more rare keywords appearing in tuples.

(3) The **Affinity** of the tuples containing the thematic keywords to the t^{DS} (recall, that as explained in Section 2.1 [5], for each relation of a G^{DS} an affinity score is calculated). For instance, if the keywords are found within one hop from t^{DS}, they should certainly have higher impact in the ranking of the OS, than if found in tuples far from t^{DS}. For instance, the keyword “mining” found in an author's paper should have higher weight than a keyword found in a paper cited by a paper of the author. As shown in [5], affinity is a more accurate measure compared to distance (which is used by other ranking paradigms in relational databases).

Based on the above ranking criteria the thematic ranking score of an OS O (which contains q_1 in its DS tuple t^{DS}) can be calculated using the following two constituent factors:

$$score_1(O, q_1) = Im(t^{\text{DS}}), \tag{1}$$

$$score_2(O, q_2) = \frac{\sum_{t \in O} s(t, q_2)}{1 - \alpha + \alpha \cdot \frac{dl(O)}{avdl(OS)}}, \tag{2}$$

where

$$s(t, q_2) = \sum_{w \in t \cap q_2} (1 + \ln(1 + \ln(tf_w(t)))) \cdot \ln(idf_w) \cdot li(t), \tag{3}$$

$$idf_w = \frac{N_{OS} + 1}{df_w(OS)}, \tag{4}$$

$$li(t) = Af(t) \cdot Im(t). \tag{5}$$

The first factor $score_1(O, q_1)$ captures the importance of t^{DS} itself, while the second factor $score_2(O, q_2)$ captures the relevance of the OS to the keywords in q_2 (i.e., the importance and affinity of the tuples in O containing these keywords). $score_1(O, q_1)$ is simply defined by $Im(t^{\text{DS}})$, which is the *global importance* of tuple t^{DS} in the database, according to authority measures such as ObjectRank [26] and ValueRank [4]. To define $score_2(O, q_2)$, we consider each tuple $t \in O$ that includes at least one keyword from q_2. Note that the same tuple t in a relation R_i can appear multiple times in an OS (e.g., the same author can appear as a co-author in many papers in the same OS tree). Thus, $score_2(OS, q_2)$ considers such tuple instances as distinct (i.e., different) tuples in the OS. For these tuples, we compute a $s(t, q_2)$ by aggregating for each q_2-keyword w in t (i) the normalized TF-IDF score of w in t and (ii) the *local importance* of t in the OS (i.e., the affinity-weighted global importance of t). Specifically, $tf_w(t)$ is the term frequency of w in t, $li(t)$ is the local importance of t in the context of an OS, $Af(t)$ is the affinity of the relation that t belongs to, and $Im(t)$ is the global Importance of t. idf_w is the inverse document frequency of w in the database, defined by dividing the total number of OSs N_{OS} by the number of OSs

$df_w(OS)$ containing the term w. In Eq. (2), each $s(t, q_2)$ is normalized to consider the ratio between the size in words of the OS, $dl(O)$, and the average size of all OSs, $avdl(OS)$, in order not to favor large OSs; α is a tuning parameter (default to 0.5 as in [10]) used in this normalization. Observe that $score_2(O, q_2)$ will be 0 if O does not include any thematic tuples (since all tuples in the given OS will have $s(t, q_2) = 0$). Note that $score_2$ is a natural adaptation from previous work [10] that considers IR scores in ranking R-KwS results; $li(t)$ (i.e. $Af(t)*Im(t)$) has also been used in previous work [12] following the same reasoning that faraway tuples in an OS should be penalized when selecting a size-l OS synopsis.

The Combined Formula: Finally, the two scoring factors are combined as a product (i.e., in accordance to earlier work [10,11]) as to derive overall score of each OS (relevant to q_1), with respect to the thematic keywords q_2:

$$score(O, Q) = score_1(O, q_1) \cdot score_2(O, q_2) \tag{6}$$

4. Preprocessing

Given a query $Q = \langle q_1, q_2 \rangle$, our goal is to rank the OSs that have as data subject a tuple t^{DS} that contains q_1, based on their scores as defined by Eq. (6), and return the top-k OSs with the highest scores. During search, our goal is to avoid computations as far as possible. Since the data are mostly static, we invest in pre-computations in order to reduce the search cost. In the following we describe the pre-computation techniques that we apply and the information that we save from them, which is then used during search.

OS-independent factors: There are several factors in the scoring formula which are independent of the respective OS and the query keywords, but they are only related to the database tuples. These factors can be easily and cheaply indexed and obtained from the database (same indexing was also used in [12]). Specifically, the global importance $Im(t)$ for each tuple t is pre-computed and stored based on the underlying applications. Similarly, $Af(t)$ can easily be obtained after having preprocessed the affinity of the relation in G^{DS} where t belongs to. For example, in the G^{DS} shown in Fig. 2, the affinities of all relations have been preprocessed. N_{OS} can be easily calculated by the cardinality of R^{DS} (e.g., $|R^{Author}|$ for the Author OSs).

OS-dependent factors: Each relation of the database can play the role of R^{DS}, i.e., the relation from which the data subjects are obtained. As discussed above, for each candidate R^{DS}, we can generate the corresponding G^{DS} and compute the affinities between R^{DS} and all other relations in its G^{DS}. We go one step further and pre-compute the object summaries for all $t^{DS} \in R^{DS}$. However, we do not keep these OSs, because they are too expensive to store (each of them may involve thousands of tuples). Instead, we only use these OSs to compute a small number of statistics that facilitate the computation of scores of the OSs. Specifically, we compute $avdl(OS)$ (i.e., the average size of all OSs), $df_w(OS)$ (i.e., the number of occurrences of each keyword w in object summaries, used to calculate $idf_w(OS)$) for each possible keyword w, and $dl(O)$, which is the size of each OS O. In other words, we keep for each tuple in the database (which could potentially be a t^{DS}), the size of the corresponding object summary. In addition, for each thematic relation R_j^{Th} of the G^{DS}, we index for each t^{DS} the *total* number of times that it can join with the tuples in R_j^{Th}, denoted as $M^{R_j^{Th}}(t^{DS})$. We also index for each t^{DS}, the *maximum* number of times it can join with the same tuple from R_j^{Th}, denoted as $m^{R_j^{Th}}(t^{DS})$. We simply denote $M^{R_j^{Th}}(t^{DS})$ and $m^{R_j^{Th}}(t^{DS})$ as M and m respectively for simplicity when there is no ambiguity in the context. Finally, we adopt a reachability index, where for each $t^{DS} \in R^{DS}$ and for each R_j^{Th} in the corresponding G^{DS}, we keep the set of tuples in R_i for which there is join path to t^{DS}. Since such an exact reachability index is too expensive to store (quadratic to the number of tuples in the database), we resort to an approximate solution utilizing *bloom filter* (BF) sketches [27]. A BF is a space and time efficient probabilistic data structure that can be used to test whether an element is a member of a set. In our case, BF sketches store reachability relationships between tuples approximately (i.e., they include false positives). The reason for storing these additional statistics and the BFs will become apparent in Section 5.3, where we will describe our search algorithm.

5. Approaches

After having preprocessed and indexed the factors that are involved in the scoring of an OS w.r.t. a query Q, a brute-force method to solve our ranking problem is to compute all complete OSs that include the identifying keywords q_1 and then rank them by their scores. However, such an approach requires computing all the OSs at their entirety, which is an expensive process. In this section, we propose an alternative approach, which avoids the computation of the OSs. Namely, we re-formulate the problem to a kGBJ, i.e. we compute only the join paths between the data subject tuples (i.e., those that contain the identifying keywords q_1) against the thematic tuples (i.e., those that contain the thematic keywords q_2). In addition, since our objective is to find only the top-k OSs with the highest scores, we propose an optimized search technique that avoids computing join paths where possible, if these do not contribute to the top-k OSs. For this purpose we use the aforementioned preprocessed information to derive effective bounds. For the ease of discussion, we will first assume that there is a single R^{DS} (where the DS stem from) and a single R^{Th} (i.e., all tuples t that include keywords from q_2 are in a single relation). In Section 6, we will deal with the more general case where R^{Th} consists of more than one relations.

5.1. Problem reformulation

To compute the score of an OS (see Section 3), we observe that the computation of the entire OS is not necessary; we only need to know $dl(O)$ and the thematic tuples that link to t^{DS} via join paths. Consider the scenario where a user, who is looking for scholars

whose names contain “Chen” and are experts in data mining or database, submits a query $Q = \langle\{\text{“Chen”}\}, \{\text{“mining”, “database”}\}\rangle$. Given the corresponding G^{DS} of Author in Fig. 2, the naive approach will compute all the *Chen* OSs at their entirety and calculate the scores while traversing the OS trees. This way we may retrieve thematic tuples containing neither “mining” nor “database”, which do not contribute to the final scores of the respective OSs. Such IO/CPU time waste is more significant when the nodes in the G^{DS} have a large fan-out (which leads to a great number of thematic tuples), as the thematic keywords usually do not have a very high selectivity.

Therefore, we reformulate our OS ranking problem as a *top-k Group By join* problem (*k*GBJ) that takes as input a relation R^{DS}, which includes the t^{DS} of all OS that have to be ranked, and one or more relations R^{Th} in the G^{DS} of R^{DS}, which include tuples t that contain any keyword $w \in q_2$. *k*GBJ first selects only the *candidate* data subject tuples $t^{DS} \in R^{DS}$, denoted as $R^{DS}(q_1)$, such that t^{DS} contains q_1 and only the related thematic tuples $t \in R^{Th}$, denoted as $R^{Th}(q_2)$, such that any keyword $w \in q_2$ is contained in t. *k*GBJ then joins the two sets of tuples using the G^{DS} relation graph and *groups* the join pairs by t^{DS}; this way we can obtain all factors that we need to compute the score of each OS. Finally, the k OSs with the highest scores are the output.

5.2. Baseline: Bi-Directional (BD) approach

A baseline approach for solving the *k*GBJ problem is to follow the approach of a conventional database engine. Specifically, two selection operations are first applied on R^{DS} and R^{Th} to select only the set of tuples $R^{DS}(q_1)$ and $R^{Th}(q_2)$ in them that include keywords in q_1 and q_2, respectively. Then, the results of the selections are joined via the join path that connects R^{DS} and R^{Th} in G^{DS}. Since, in general, a bushy join evaluation plan would be used, which would minimize the number of intermediate results produced, we call this approach *Bi-Directional* (BD) *k*GBJ evaluation. The bi-directional strategy has also been used in early work on R-KwS [28].

In particular, BD first computes the join path (JP) from G^{DS} that connects R^{DS} to the thematic relation R^{Th}. For example, for query $Q = \langle\{\text{“Chen”}\}, \{\text{“mining”, “database”}\}\rangle$, suppose the DS relation and the thematic relation are R^{Author} and $R^{Conference}$ respectively, as shown in Fig. 2, the corresponding join path is R^{Author}–R^{Paper}–$R^{ConfYear}$–$R^{Conference}$.

Then, BD estimates the optimal meeting point in a bushy evaluation plan that has $R^{DS}(q_1)$ and $R^{Th}(q_2)$ at its two ends. In other words, the evaluation plan considered consists of a left-deep plan, starting by joining $R^{DS}(q_1)$ with its next relation in the JP and continuing this way and a right-deep plan, starting by joining $R^{Th}(q_2)$ with its previous relation in JP and continuing this way. Eventually, the two paths are joined at a *meeting point*. As in a query optimizer, given the sizes of $R^{DS}(q_1)$ and $R^{Th}(q_2)$, the estimation of the optimal meeting point is done with the help of statistics; specifically, by knowing the expected number of join results between *neighboring* relations in our relational schema, we can estimate the costs and number of results for each pair of relations to be joined and we can select the meeting point that minimizes the total join cost of BD. Fig. 4 illustrate 3 possible meeting points (as root nodes) of the aforementioned query. Note that this heuristic does not consider all the possible joining plans, which are of exponential scale w.r.t. the length of the join path.

After computing the join, the results are grouped by t^{DS}; i.e., for each distinct $t^{DS} \in R^{DS}(q_1)$, for each join result that includes t^{DS}, the tuple $t \in R^{Th}$ included in the result contributes to $score_2(O, q_2)$, where O is the OS of t^{DS} (see Eq. 2). All other factors required to complete the computation of $score(O, Q)$ (i.e., $dl(O)$, $avdl(OS)$, etc.) are obtained from the precomputed data as discussed in Section 4. Finally, the OSs are ranked based on their scores and the k ones with the highest scores are output.

5.3. Top-k BD (kBD) approach

The rationale of this algorithm is to avoid the entire BD traversal and processing of our input (i.e. of $R^{DS}(q_1)$ and $R^{Th}(q_2)$). Our algorithm achieves this by estimating upper and lower bounds for each OS and by managing them in descending order of their upper bounds in a max-heap. Then, we iteratively process the OSs with the largest upper bound by tightening bounds until we produce the top k OSs. Hereby, we firstly discuss OSs upper and lower bounds estimations and then their top-k generation.

5.3.1. Upper and lower bounds of OSs

We firstly define an initial score of an OS O that disregards the contributions of any thematic tuples as follows:

$$In(O) = Im(t^{DS}) \cdot \frac{1}{1 - \alpha + \alpha \cdot \frac{dl(O)}{avdl(OS)}} \tag{7}$$

where $avdl(OS)$ and $dl(O)$ can be pre-calculated as discussed in Section 4. Then, we have

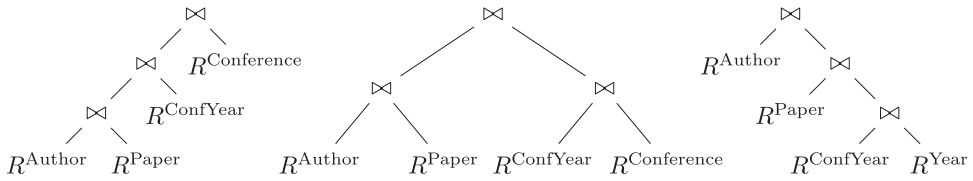

Fig. 4. Meeting Points Example.

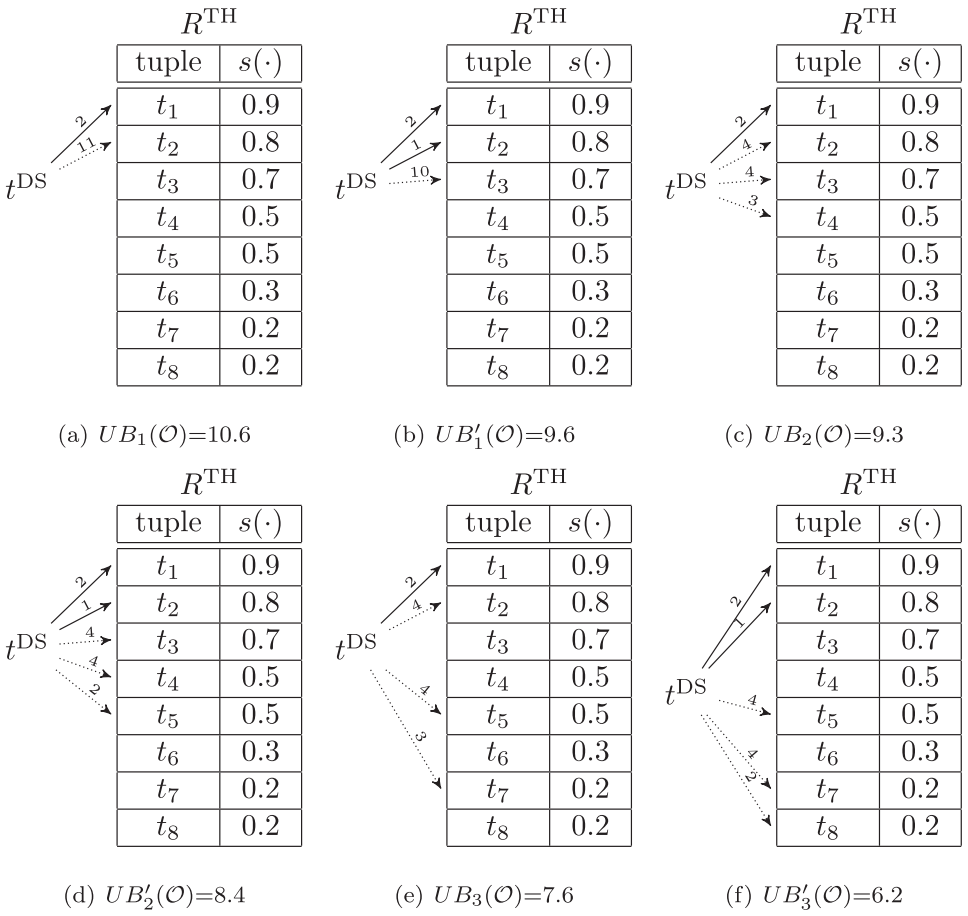

Fig. 5. Calculating the Upper Bound Scores of an OS O ($In(O)$ = 1.0, M = 13, m = 4).

$$score(O, Q) = In(O) \cdot \sum_{t_j \in R^{\mathrm{Th}}(q_2)} n_j \cdot s(t_j, q_2) \tag{8}$$

where n_j denotes the number of times a thematic tuple t_j appears in OS O. Let $N = \sum_{t_j \in R^{\mathrm{Th}}(q_2)} n_j$ and let $M = \sum_{t_j \in R^{\mathrm{Th}}} n_j$ (as defined in Section 4). Obviously, $R^{\mathrm{Th}}(q_2) \subseteq R^{\mathrm{Th}}$, hence $N \leq M$. Therefore, $score(O, Q)$ is upper-bounded by $In(O) \cdot M \cdot s(\hat{t}, q_2)$, where $\hat{t} \in R^{\mathrm{Th}}(q_2)$ is the tuple with the largest $s(\cdot)$ score. We denote $s(t, q_2)$ as $s(t)$ for simplicity when there is no ambiguity in the context.

Specifically, let us assume that all the thematic tuples in $R^{\mathrm{Th}}(q_2)$ are *sorted* in descending order of their $s(\cdot)$ scores, and that t_i is the i-th tuple after sorting. Suppose that we have examined the first l thematic tuples in R^{Th} so far. Then, $LB(O)$ is a lower bound of the final score of O and defined as

$$LB(O) = In(O) \cdot \sum_{j=1}^{l} n_j \cdot s(t_j). \tag{9}$$

Let $\sigma = \sum_{j=1}^{l} n_j$; we can easily derive that $score(O, Q)$ is upper bounded by $UB_1(O, Q)$, which is defined as

$$UB_1(O, Q) = LB(O) + In(O) \cdot (M - \sigma) \cdot s(t_{l+1}). \tag{10}$$

Example 1. Let $In(O)$ = 1.0, M = 13 and m = 4, Fig. 5(a) shows an example on how to calculate $UB_1(\cdot)$. Here, we found 2 actual joins between O and t_1 (depicted with a solid arrow) and thus $LB(O)$ = 1.8. Therefore, we assume all the remaining possible joins $M - \sigma$, 13 − 2 = 11 (depicted with dotted arrows) occur between O and t_2, since t_2 is the tuple with the largest $s(\cdot)$ score among all other tuples in $R^{\mathrm{Th}}(q_2)$. Therefore, $UB_1(O)$ = 1.8 + 1.0.8 = 10.6. Fig. 5(b) illustrates the updated $UB_1(\cdot)$ score after we find another join between O and t_2.

Although $UB_1(\cdot)$ is a correct upper bound of $score(O, Q)$, it is not a tight upper bound as we assume that all the remaining possible joins are of score $s(t_{l+1})$, which correspond to the tuple with the next largest score. As we explained in Section 4, for each data subject t^{DS}, we also index m, the maximum number of times a thematic tuple in R^{Th} can join with t^{DS}. This information can be utilized to obtain a tighter bound by assuming that each tuple in $R^{\mathrm{Th}}(q_2)$ can join only up to m times with t^{DS} and thus consider possible joins

with tuples with smaller score than that of score $s(t_{l+1})$. Let $\gamma = \min\{m\cdot(|R^{\mathrm{Th}}(q_2)| - l), M - \sigma\}$ be the total number of joins that can be found between t^{DS} and all the remaining thematic tuples and let $d = \lfloor \gamma/m \rfloor$ and $r = \gamma \bmod m$. Then, we can define a tighter upper bound as follows:

$$UB_2(O, Q) = LB(O)) + In(O)\cdot\left(\sum_{j=1}^{d} s(t_{l+j})\cdot m + s(t_{l+d+1})\cdot r\right) \tag{11}$$

Example 2. Fig. 5(c) shows an example on how to calculate $UB_2(\cdot)$. We have two actual joins with t_1, thus we have 11 remaining joins to assign in order to obtain an upper bound. Since $m = 4$, we distribute the 11 joins to the remaining thematic tuples as follows: we assign 4 joins to tuples t_2 and t_3 and the remaining 3 joins to t_4. Therefore, $UB_2(O) = 1.8 + 1.0 \times 4 \times 0.8 + 1.0 \times 4 \times 0.7 + 1.0 \times 3 \times 0.5 = 9.3$. Fig. 5(d) illustrates the updated $UB_2(\cdot)$ score after we find another one join between O and t_2.

In addition, with the help of bloom filters (BF), we can further tighten the aforementioned upper bound by considering only actual joins between t^{DS} and thematic tuples. In particular, in Eq. (11) we assume that t^{DS} can join m times with each of the remaining thematic tuples, until it reaches the total amount of M joins or the end of the thematic list. However, if a tuple pair (t^{DS}, t_j) does not pass the BF test, then we know for sure that t^{DS} cannot join with t_j via the given join path. Hence, we can safely skip t_j and consider the next thematic tuple t_{j+1}. Note that, the use of BF does not affect correctness as they are only used in order to obtain a closer bound. Assume that the sequence of tuples satisfying the BF test are $t_{a_1}, t_{a_2}, \cdots, t_{a_{d+1}}$ and the corresponding upper bound of joins is m, except for the last tuple where we assume we have r joins. Then, we can estimate a tighter bound as follows:

$$UB_3(O, Q) = LB(O) + In(O)\cdot\left(\sum_{j=1}^{d} m\cdot s(t_{a_j}) + r\cdot s(t_{a_{d+1}})\right). \tag{12}$$

It is not difficult to verify that $UB_3(\cdot)$ is the tightest bound of all the aforementioned upper bounds, i.e., $UB_1(\cdot) \geq UB_2(\cdot) \geq UB_3(\cdot)$. Note that hereafter we use $UB_3(\cdot)$ when we refer to the upper bound score of an OS.

Example 3. Fig. 5(e) shows an example on how to calculate $UB_3(\cdot)$. We have $LB(O) = 1.8$ and we still have 11 possible joins according to M. In contrast to the previous cases, we know that O cannot join with t_3, t_4 and t_6, which means that we can disregard these tuples while estimating bounds. Analogously, we can distribute the 11 joins as follows: 4 joins to tuples t_2 and t_5 and the remaining 3 joins to t_7. Therefore, $UB_3(O) = 1.8 + 1.0 \times 4 \times 0.8 + 1.0 \times 4 \times 0.5 + 1.0 \times 3 \times 0.2 = 7.6$. Fig. 5(f) illustrates the updated $UB_3(\cdot)$ score after we find an additional join between O and t_2.

5.3.2. Top-k Bi-Directional (kBD) Algorithm

Hereby, we describe how the kBD algorithm computes the kGBJ results using the proposed bound estimation method (Algorithm 1). First, kBD algorithm sorts all the thematic tuples in $R^{\mathrm{Th}}(q_2)$ in descending order of their $s(\cdot)$ scores (according to Eq. 3) and generates the lower bounds $LB(\cdot)$ and upper bounds $UB(\cdot)$ for all the DS tuples in $R^{\mathrm{DS}}(q_1)$.

Algorithm 1. kBD Algorithm

```
    kBD (R^DS(q1), R^Th(q2), k)
1:  H:=∅;
2:  L^Th:=R^Th(q2);
3:  sort tuples in L^Th in descending order of their s(·) scores;
4:  for each O w.r.t. t^DS in R^DS(q1) do
5:     LB(O):=0; UB(O):= CALCUB O;
6:     insert O into H with priority UB(O);
7:  while k > 0 ∧ H is not empty do
8:     pop O_cur from H;
9:     O_next:=H.top();
10:    if LB(O_cur) ≥ UB(O_next) then
11:       report O_cur as a result;
12:       k:=k − 1;
13:    else
14:       t_l:= the next tuple in L^Th can join with O_cur;
15:       n := JOIN O_cur, t_l;
16:       LB(O_cur):= LB(O_cur) + n·In(O_cur)·s(t_l, q2);
17:       UB(O_cur):= CALCUB O_cur;
18:       push O_cur back into H with priority UB(O_cur);
```

OS	$UB(.)$	$LB(.)$
$\mathcal{O}_1$	8.0	0
$\mathcal{O}_2$	6.0	0
$\mathcal{O}_3$	10.0	0
$\mathcal{O}_4$	5.0	0
$\mathcal{O}_5$	4.0	0

$H=\langle\mathcal{O}_3,\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_4,\mathcal{O}_5\rangle$, $\mathcal{O}_{cur}=\mathcal{O}_3$, $\mathcal{O}_{next}=\mathcal{O}_1$

(a) Initialization

OS	$UB(.)$	$LB(.)$
$\mathcal{O}_1$	8.0	0
$\mathcal{O}_2$	6.0	0
$\mathcal{O}_3$	7.0	6.5
$\mathcal{O}_4$	5.0	0
$\mathcal{O}_5$	4.0	0

$H=\langle\mathcal{O}_1,\mathcal{O}_3,\mathcal{O}_2,\mathcal{O}_4,\mathcal{O}_5\rangle$, $\mathcal{O}_{cur}=\mathcal{O}_1$, $\mathcal{O}_{next}=\mathcal{O}_3$

(b) Iteration 1

OS	$UB(.)$	$LB(.)$
$\mathcal{O}_1$	7.5	5.0
$\mathcal{O}_2$	6.0	0
$\mathcal{O}_3$	7.0	6.5
$\mathcal{O}_4$	5.0	0
$\mathcal{O}_5$	4.0	0

$H=\langle\mathcal{O}_1,\mathcal{O}_3,\mathcal{O}_2,\mathcal{O}_4,\mathcal{O}_5\rangle$, $\mathcal{O}_{cur}=\mathcal{O}_1$, $\mathcal{O}_{next}=\mathcal{O}_3$

(c) Iteration 2

OS	$UB(.)$	$LB(.)$
$\mathcal{O}_1$	6.2	6.0
$\mathcal{O}_2$	6.0	0
$\mathcal{O}_3$	7.0	6.5
$\mathcal{O}_4$	5.0	0
$\mathcal{O}_5$	4.0	0

$H=\langle\mathcal{O}_3,\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_4,\mathcal{O}_5\rangle$, $\mathcal{O}_{cur}=\mathcal{O}_3$, $\mathcal{O}_{next}=\mathcal{O}_1$

(d) Iteration 3

Fig. 6. The *k*BD Algorithm for $k = 1$.

.

Then, it prioritizes the OSs w.r.t. their $UB(\cdot)$ scores using a max-heap H, and select the OS with the largest upper bound score to process as it is the most promising DS to appear in the result top-k set. We denote the selected OS as O_{cur}. Let the next OS be the one that has the second largest $UB(\cdot)$ score in H, denoted as O_{next}. If $LB(O_{cur}) \geq UB(O_{next})$, then O_{cur} is guaranteed to be the best DS among those in the heap H, hence it is pop out and reported as a result. Otherwise we calculate the amount of joins of O_{cur} with its next thematic tuple and update accordingly the lower bound $LB(O_{cur})$ and upper bound $UB(O_{cur})$. *k*BD terminates as soon as k DSs are reported, or there are no more candidates in H.

The correctness of this branch-and-bound approach can be verified by the fact, that the algorithm outputs the OS, O_{cur}, with $LB(O_{cur}) \geq UB(O_{next})$. Where O_{next} is the next OS in the Heap H; recall that H is ranked according to the upper bound score of OSs. Thus, the returned OS will always have a score larger than (the upper bound of) all remaining OSs in the heap H.

Example 4. Fig. 6 illustrates a running example of *k*BD with 5 OSs and $k = 1$. In the beginning, O_3 is the OS with the largest $UB(\cdot)$ score, therefore O_3 is chosen as the current and O_1 as the next OS. Since $LB(O_3) < UB(O_1)$, we process O_3 with its next thematic tuples, and update the corresponding upper bound and lower bound score to 7 and 6.5 respectively (Fig. 6 (b)). In the next round, O_1 becomes the most promising OS. We select O_1 as the current and implement the joins between O_1 and its next thematic tuple, which derives $LB(O_1) = 7.5$ and $UB(O_1) = 5.0$ (Fig. 6 (c)). Then, O_1 is selected again due to its highest $UB(\cdot)$ score among all those in H; the upper bound and lower bound score of O_1 is updated to 6.2 and 6.0 respectively (Fig. 6 (d)). Finally, O_3 comes into the top of the heap, and since $LB(O_3) > UB(O_1)$ (O_1 is the next promising OS in this round), O_3 is guaranteed to be the best OS in H. As a result, O_3 is reported and *k*BD terminates since we have found the top-1 best OS.

We now elaborate on the details of the two routines in Algorithm 1: JOIN and CALCUB.

JOIN (O_{cur}, t_l). This function finds the actual amount of joins between O_{cur} and t_l (after succeeding the BF test) by implementing the join. We adopt the same strategy for implementing joins (between the DS tuples and thematic tuples) as in the BD algorithm (Section 5.2); i.e., we use a *meeting point*. Then, we implement the joins from both sides, i.e. from the DS tuple and the thematic tuple sides towards the meeting point relation and merge the two results instances. Note that one DS (or thematic) tuple can be chosen by the *k*BD algorithm more than once. Therefore, we cache the join result instances of each DS (or thematic) tuple at the meeting point as to avoid re-computation.

CALCUB (O_{cur}). This function calculates the $UB_3(\cdot)$ score, which is the tightest upper bound we estimate by using the BF index. A

naive update operation of $UB_3(\cdot)$ score is to assign the remaining number of joins to those unseen tuples from scratch. However, this method may query the same pair several times from the BF index. For example, in Fig. 5(e), the tuple pair (t^{DS}, t_5) is tested when computing the initial upper bound score. In the next round, after we verify the tuple pair (t^{DS}, t_2) by finding the actual joins, the same pair (t^{DS}, t_5) will be checked again if we calculate the upper bound score from the very beginning. In view of this, we propose to maintain a list of thematic tuples that have been processed so far against the BF index for each DS tuples. Each time when a thematic tuple is verified, we update the list incrementally. For instance, in Fig. 5(e), at first we have a list of thematic tuples $\{t_1, t_2, t_5, t_7\}$. After verifying t_2, we have 3 remaining joins since t_2 can join with t^{DS} only once. Therefore, we add one more join on the last tuple in the list, i.e. t_7 and the remaining 2 joins to t_8, assuming that the pair (t^{DS}, t_8) pass the bloom filter test. In general, the incremental maintenance of thematic tuples can be considered as a sliding window of size M on R^{Th}; where each pair of t_i and t^{DS} tuples that passes the BF test has size m or 0 otherwise (i.e. they fail the BF test).

6. Multiple thematic relations

Thematic keywords are very likely to be found in more than one relations of the G^{DS}. In this section we show how to extend the solutions in Section 5, i.e. BD and kBD, in order to solve the general case where more than one thematic relations are involved.

6.1. Baseline: Holistic BD (HBD) algorithm

Assume that there are j thematic relations in the data subject graph G^{DS}, denoted as R_1^{Th}, …, R_j^{Th}. A straightforward approach is to extend the BD algorithm by running the BD algorithm j times, i.e. one for each thematic relation. We denote this method as the Holistic Bi-Directional Algorithm (HBD).

For example, consider the Author G^{DS} (Fig. 2) and assume that we have 3 thematic relations, $R_1^{Th} = R^{Conference}$, $R_2^{Th} = R^{PaperCitedby}$ and $R_3^{Th} = R^{PaperCites}$. Then, HBD will implement the three corresponding join paths, (i.e. $\mathcal{JP}_1$, $\mathcal{JP}_2$ and $\mathcal{JP}_3$ of Table 1) for R_1^{Th}, R_2^{Th} and R_3^{Th} respectively and will aggregate the score for each t^{DS}. Finally, HBD will output the top-k OSs.

6.2. Holistic Top-k BD (HkBD) algorithm

Given j thematic relations R_1^{Th}, …, R_j^{Th}, we can extend analogously the original k BD algorithm by defining appropriate upper and lower bound scores for each DS. We can easily see that the sum of the upper (resp. lower) bound scores of all join paths is the upper (resp. lower) bound score of an OS, namely

$$UB^H(O, Q) = \sum_{\mathcal{JP}_i} UB^{\mathcal{JP}_i}(O, Q), \tag{13}$$

$$LB^H(O, Q) = \sum_{\mathcal{JP}_i} LB^{\mathcal{JP}_i}(O, Q), \tag{14}$$

where $\mathcal{JP}_i$ ranges over all thematic paths and $UB^{\mathcal{JP}_i}(\cdot)$ (resp. $UB^{\mathcal{JP}_i}(\cdot)$) is the upper (resp. lower) bound score of O as calculated by Eq. (12) (resp. Eq. (9)) w.r.t. the join path $\mathcal{JP}_i$.

In contrast to the previous scenario where we had only one thematic relation, in this multiple thematic scenario, each DS can now join with tuples from a set of thematic relations (i.e. from R_1^{Th}, …, R_j^{Th}). Thus, we need a selection policy that will suggest from which thematic relation to choose tuples to process next (i.e. to verify for actual joins against O_{cur} and update lower and upper bounds for the O_{cur}, lines 15–17 of Algorithm 1). We suggest to choose and process a tuple from the thematic relation which we consider as the most promising for the k GBJ. We consider as the most promising thematic relation the one that its next tuple can give the largest $m^{R_i^{Th}}(t^{DS})\cdot s(\cdot)$ score w.r.t. query Q. The rationale is that by picking the thematic tuple with the largest $m^{R_i^{Th}}(t^{DS})\cdot s(\cdot)$ score, denoted as t_{next}^{Th}, according to formulas (9) and (12), we will either (1) decrease $LB^H(\cdot)$ further by finding new valid joins between O_{cur} and t_{next}^{Th}; (2) or decrease the $UB^H(\cdot)$ further. Hence, either case will result in an earlier termination of the top-k algorithm.

According to our algorithm (see Algorithm 1) in line 15, we need to verify the joins of the identifying against thematic tuples as to obtain their actual number of joins. We observe that in most cases, some of these join paths actually share a common prefix and thus we can exploit this property by avoiding processing these join paths independently. Thus, in this algorithm we suggest to force the meeting point to the common G^{DS} prefix which is shared by all paths, (e.g., R^{Paper} for the $\mathcal{JP}_1 - \mathcal{JP}_3$ example). Then, we can compute the join result at R^{Paper} once and reuse it later for the other paths.

On the other hand, according to the baseline *HBD* approach, the meeting point for the two join paths in our example ($\mathcal{JP}_1$ and

Table 1
Examples of Join Paths.

ID	Path Name	20%|DS|	20%|Th|
$\mathcal{JP}_1$	DBLP: Author-Paper-ConfYear-Conference	68 K	593
$\mathcal{JP}_2$	DBLP: Author-Paper-PaperCitedBy	68 K	103 K
$\mathcal{JP}_3$	DBLP: Author-Paper-PaperCites	68 K	103 K
$\mathcal{JP}_4$	TPC-H: Cust.-Ord.-Lineitem-Partsupp-Part	30 K	400 K
$\mathcal{JP}_5$	TPC-H: Cust.-Ord.-Lineitem-Partsupp-Supplier	30 K	K

$\mathcal{JP}_2$) will be R^{ConfYear} and R^{Paper} respectively. Thus, according to *HBD*, we need to cache the join-able tuples up to R^{ConfYear} relation for the first path and join-able tuples up to R^{Paper} for the second path. However, when we are computing the join results for the second path, we cannot reuse the results from the first join cache, namely we have to re-compute the join starting from t^{DS} again. Thus, the H*k*BD approach is advantageous in this aspect over the HBD algorithm, as it facilitates reuse of join results.

7. Experimental evaluation

In this section, we experimentally evaluate the proposed thematic ranking of OSs and algorithms. Section 7.1 discusses the implementation details, datasets and other experimental setups. Section 7.2 thoroughly evaluates the effectiveness of the proposed thematic ranking of OSs with the help of human evaluators. Section 7.3 discusses the efficiency of our proposed solutions. We do not present any comparisons against any other existing work; to the best of our knowledge, there is not any previous work that we can meaningfully compare our results with. For instance, our work is semantically (and technically) different that existing work in R-KwS and thus is not meaningfully comparable (see our thorough discussion about the differences in Section 2).

7.1. Setup

We implemented all our tested methods in C++ using a 3.40 GHz quad-core machine running Ubuntu 12.04, with 16 GB of main memory. We use two databases: DBLP and TPC-H. DBLP contains 4 relations: Author, Paper, ConfYear, and Conference (Fig. 3). The number of tuples of these relations are 341,623, 519,931, 11,588, and 2968 respectively. TPC-H includes 8 relations: Customer, Orders, Part, Supplier, Partsupp, Lineitem, Nation, and Region. The cardinality of these relations are 150,000, 1,500,000, 200,000, 10,000, 800,000, 6,001,215, 25, and 5 respectively. We used scale factor 1 in generating the TPC-H dataset.

We use affinity scores of relations as calculated in [12], (alternatively an expert can define the G^{DS} s manually, i.e., select which relations to include in each G^{DS} and their affinities). We use ObjectRank (global) [26] and ValueRank [4] to calculate the global importance for the tuples in DBLP and TPC-H databases, respectively. Moreover, the scale parameter α was set to 0.5 as in [10].

Table 2 presents the sizes of the reachability indexes that we investigate per join path. For the bloom filter index, we set the false positive probability to 0.01 and 0.001 for DBLP and TPC-H respectively. Note that, we used a smaller false positive probability for the TPC-H bloom filter since the number of elements in the bloom filter of TPC-H is much larger. We also present, for comparison, the sizes of an exact pair reachability index that stores the exact tuple pairs. In summary, we observe that the size of our BF index is relatively small (and at the same time very fast) and thus can easily fit into main memory.

7.2. Effectiveness

We measure the effectiveness of our approach on the DBLP database only (the TPC-H database due to its synthetic content was inappropriate for meaningful evaluation here). Since the DBLP database includes data about real people and their papers, we asked the DSs themselves (i.e., fifteen authors listed in DBLP) to do the thematic ranking for sets of OSs for various thematic keywords (a similar methodology was also used in [29]). None of our evaluators were involved in this paper. The rationale of this evaluation is that the DSs themselves have best knowledge of their work and of other peers in the various thematic areas and thus can provide a better judgement on such rankings. More precisely, firstly, we familiarized them with the concepts of OSs in general and their thematic ranking in particular. Specifically, we explained that an OS includes information about a particular DS and that some of their tuples also include text that can define some thematic context. Thus, the amount and location of the occurrences of such thematic words influence their ranking for the particular thematic keyword. Then, we presented sets of OSs and asked them to rank them according to the thematic keywords, e.g., a set of OSs for *Chen* authors and the thematic word "Mining". Note that for each query, e.g., ⟨{"Chen"}, {"Mining"}⟩, the result comprises only OSs that contain the thematic word (i.e., "Mining"). For instance, for the given query example, only 43 OSs out of the 1982 *Chen* OSs contain the thematic keyword "Mining". In order to further help them with their assessment we also provided them with some useful statistical information; e.g., size in tuples, words, frequency of words and their location in the OS, etc. Table 3 summarizes the queries we used. More precisely, for each identifying keyword (q_1) we combine it with all thematic keywords (q_2), thus we have in total 6*4 = 24 distinct queries. We also depict the frequency of each keyword in the database. We measure the effectiveness of our approach by using the following metrics (which have been previously used by earlier related work).

Metric 1: The amount of relevant top-1 answers (%#Rel) [10]: We compare the top-1 proposed by our evaluators against the top-1 proposed by our algorithm. For example, for query ⟨{"Chen"}, {"Mining"}⟩ both users and our approach proposed

Table 2
Reachability indexes.

Path	Bloom Filter Size	Exact Pair Size
$\mathcal{JP}_1$	1.35 MB	9.06 MB
$\mathcal{JP}_2$	1.05 MB	7.05 MB
$\mathcal{JP}_3$	1.06 MB	7.13 MB
$\mathcal{JP}_4$	10.20 MB	45.79 MB
$\mathcal{JP}_5$	10.20 MB	45.79 MB

Table 3
Summary of effectiveness queries.

Identifying Keywords (q_1)	**Frequency in DBLP**
David	4235
Chen	1982
Wang	1778
Alan	660
John	3717
Nick	179
Thematic Keywords (q_2)	**Frequency in DBLP**
Mining	2961
System	32,253
Logic	65
Data	22,500

Table 4
%#Rel and R-Rank.

%#Rel	**R-Rank**
0.73	0.85

Table 5
Precision(Recall) and ranking correlation.

k	**Precision (= Recall)**	**Ranking correlation**
5	92.0%	0.84
10	96.5%	0.92
15	98.8%	0.96
20	100%	0.98
25	100%	0.99

Ming-Syan Chen as a top-1 result. More precisely, the %#Rel of Table 4 depicts the average (as a percentage) of the relevant top-1 results of all users for all queries (e.g., if our system matches in average the relevant top-1 of 18 out of 24 queries for DBLP database per user, then %#Rel = 0.75).

Metric 2: Mean Reciprocal Rank (R-Rank) [10]: We search for the first relevant answer. Namely, R-Rank is 1 divided by the rank at which the evaluators' top-1 result is returned or 0 if it is not returned by our approach, e.g., for the above example, R-Rank would be 1 since both users and our approach proposed *Ming-Syan Chen* as a top-1 result. Similarly to %#Rel, the R-Rank of Table 4 averages the R-Rank of our evaluators' top-1s answers per queries against ours. Evidently, the R-Rank results have a strong correlation with the corresponding %#Rel results.

Metric 3: Precision/Recall: Precision is the number of relevant documents retrieved divided by the number of retrieved documents. Recall is the number of relevant documents retrieved divided by the number of relevant documents (used in [5]). More precisely, we compare the top-k relevant results proposed by evaluators and ours. Table 5 averages all evaluators precision for all queries for various values of k. Note that, in our case since the two comparative sets have equal size then precision and recall are equal (i.e. the set retrieved by our approach and the set proposed by evaluators have common size k). Note that as k increases the precision/recall results are improved.

Metric 4: Ranking Correlation.[2] Ranking correlation measures the correlation of the ranking of evaluators against our approach for various values of k (this metric has been used in [5]). More precisely, in Table 5, we average the correlation of the users' ranking with our methodology's. Here, we observe that the results of this metric have strong correlation with the precision/recall.

Discussion: The effectiveness results are very encouraging. Namely, %#Rel = 0.73, R-Rank = 0.85 whereas Precision and Recall range in 92%-100% and correlation ranges in 0.84-0.99. Another interesting observation is that as k increases precision/recall and correlation increases. Summing up, our thematic ranking model obtains very accurate results in finding the top-1 answer (metrics 1 and 2) and in finding the top-k results (metrics 3 and 4).

7.3. Efficiency

We evaluate the efficiency and scalability of our proposed algorithms, as a function of k and for various selectivities (i.e. amount of tuples) of the data subject and thematic keywords tuple sets (i.e., $|R^{DS}(q_1)|$ and $|R^{Th}(q_2)|$ respectively, also denoted as $|DS|$ and $|Th|$ for

[2] en.wikipedia.org/wiki/Rank_correlation

Table 6
Parameters.

Parameter	Values (**Default Value**)
k	1, 2, **4**, 8, 16
$\|DS\|$ (i.e., $\|R^{DS}(q_1)\|$)	5%, 10%, **20%**, 40%, 80%
$\|Th\|$ (i.e., $\|R^{Th}(q_2)\|$)	5%, 10%, **20%**, 40%, 80%

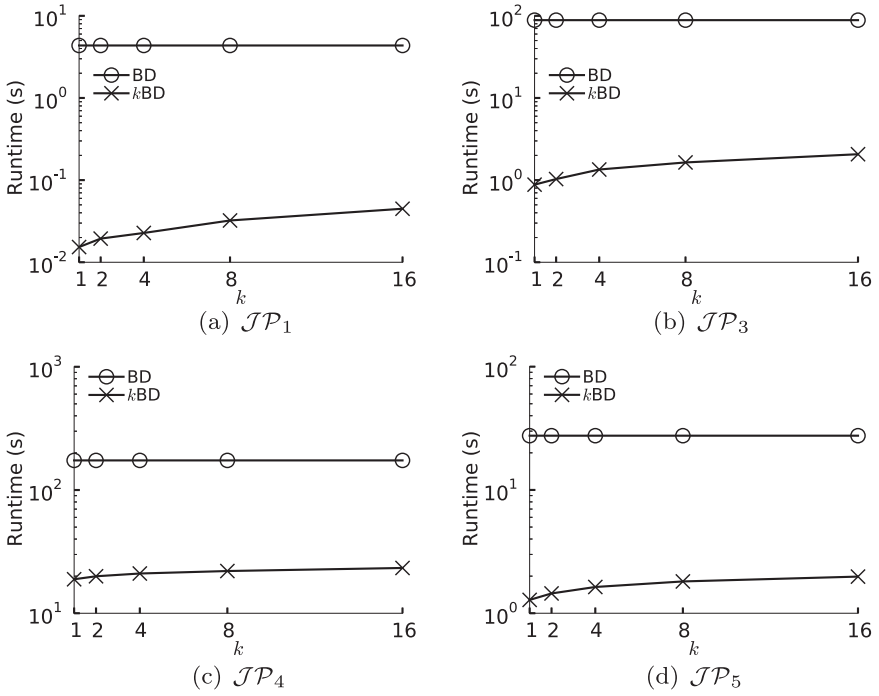

Fig. 7. Efficiency of BD and kBD for Various Values of k.

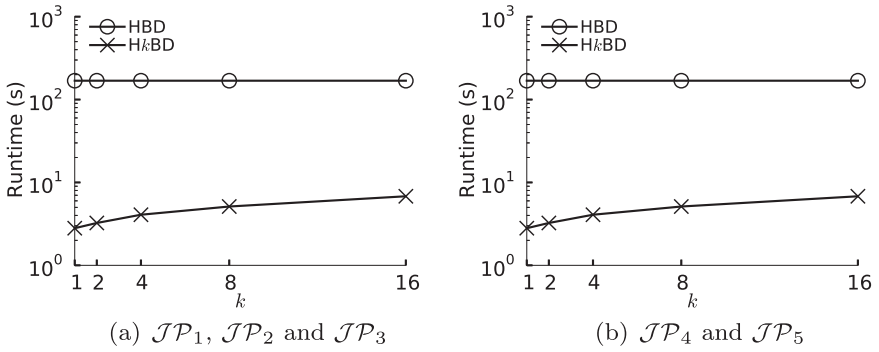

Fig. 8. Efficiency of HBD and HkBD for Various k.

short). Table 6 shows the values of the three parameters with the default values in bold font. For each selectivity setting, we select randomly four different data subject tuple sets and four different thematic tuple sets; i.e., their combination will give us in total 16(= 4 × 4) different tuple sets. We run our algorithms on each tuple set combination and obtain the average runtime.

First, we compare the performance of k BD and BD on single join paths for various values of k. We set the selectivity of the data subjects and thematic tuples to their defaults values (i.e., 20%; see Table 1). Fig. 7 shows their performance on 4 different join paths. We omit the results for $\mathcal{JP}_2$ as they are similar to that on $\mathcal{JP}_3$. First, we observe that our proposed solution (kBD) constantly outperforms the baseline method BD. We also observe, that since BD needs to conduct a complete join between data subject and thematic relations, its performance is irrelevant to k and remains the same for all values of k. Then, we also show in Fig. 8(a) and (b) the efficiency of HBD and HkBD on multiple thematic relations when k varies. We observe similar trends in the results as in the single join path case.

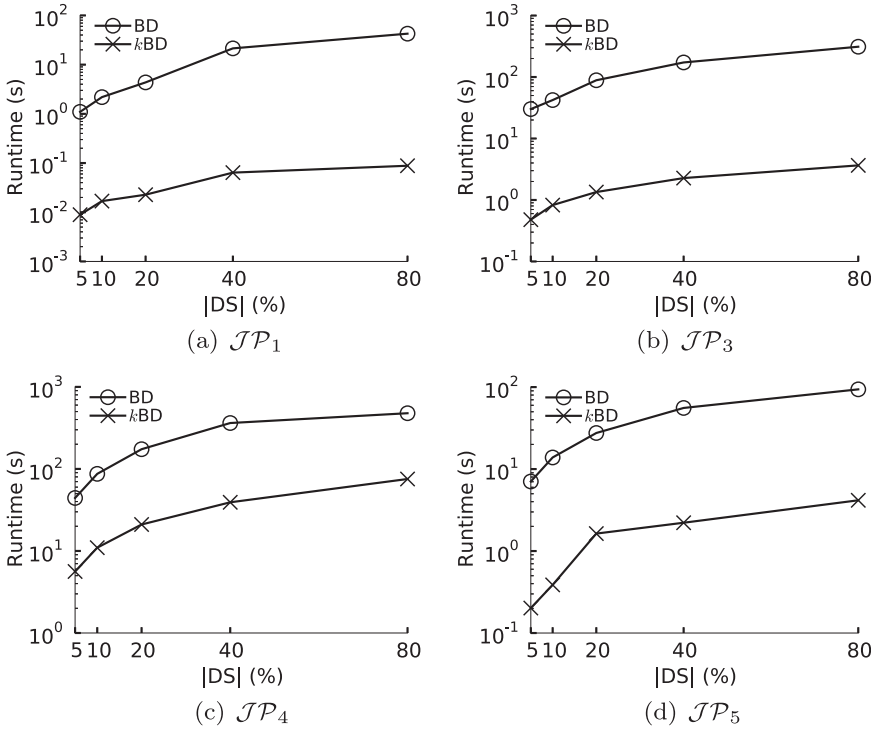

Fig. 9. Efficiency of BD and *k*BD for Various |*DS*|(%).

Scalability: Fig. 9 shows the efficiency of BD and *k*BD when the selectivity of the data subject increases; we set the selectivity of thematic tuples to 20% and *k* to 4. Note that the runtime of both methods increases as the number of data subjects increases. For instance, in the BD algorithm, increasing |*DS*| results in the increase of join checks of DS tuples against the meeting point. In addition to that, in the *k*BD algorithm, increasing |*DS*| results to more operations on H, i.e., more objects are popped and pushed in H. Fig. 10

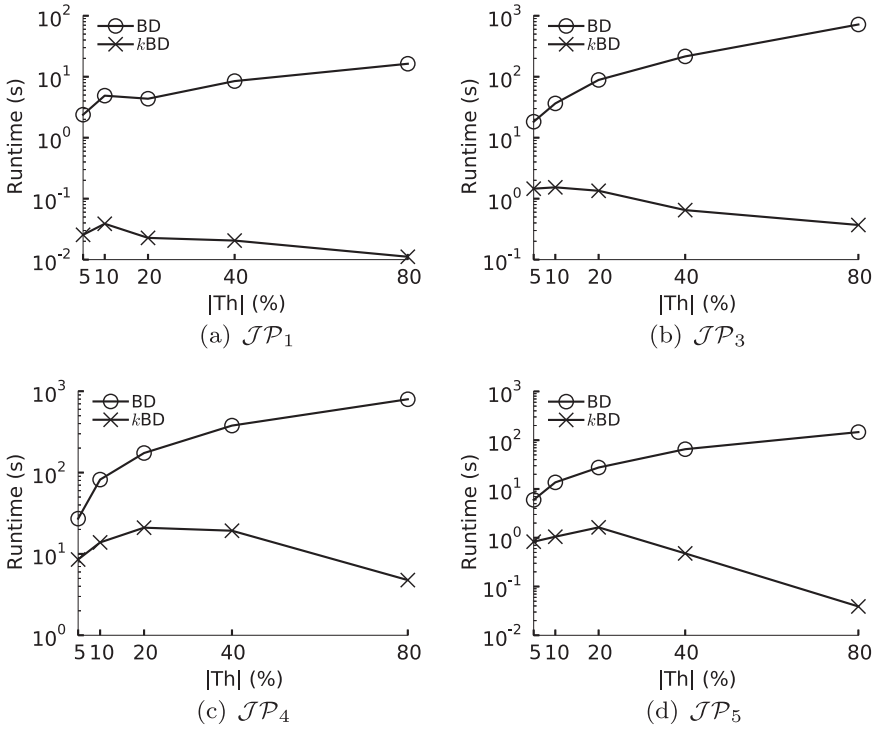

Fig. 10. Efficiency of BD and *k*BD for Various |*Th*|(%).

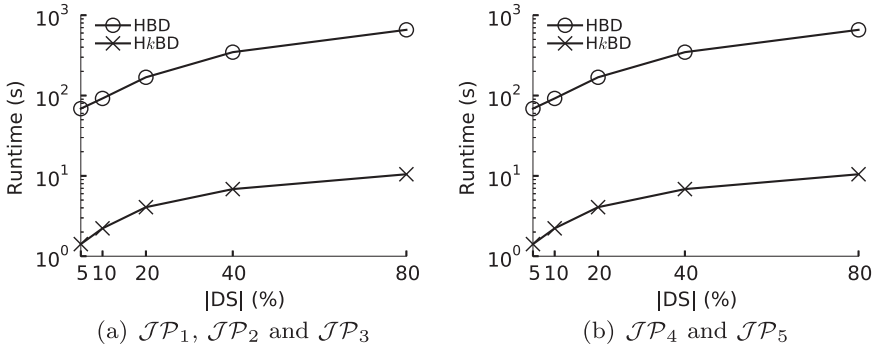

Fig. 11. Efficiency of HBD and H*k*BD for Various |*DS*|(%).

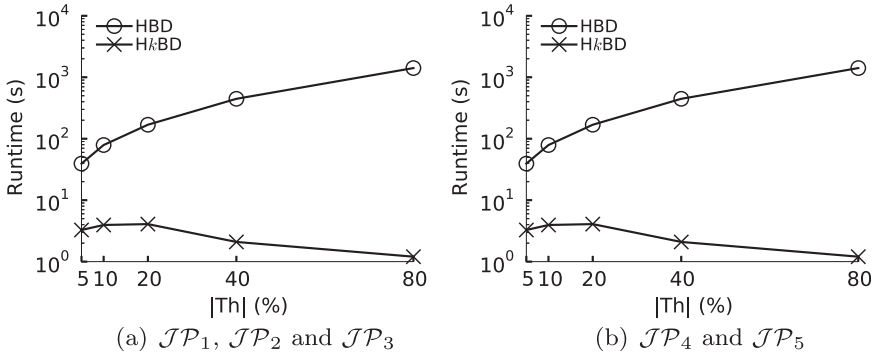

Fig. 12. Efficiency of HBD and H*k*BD for Various |*Th*|(%).

shows the performance of the algorithms when the selectivity of thematic tuples |*Th*| varies (with |*DS*| set to 20% and k to 4). For the BD, the increase of thematic tuples results in larger joins of relations and thus an increase of the runtime. For *k*BD, the runtime slightly increases when the selectivity of thematic tuples increases, since more thematic tuples need to be verified by the bloom filters. However, when the selectivity of thematic keywords becomes larger the runtime decreases. This is because, with more thematic tuples we can calculate the exact score of DSs (rather upper bounds) and thus reach the top-k condition more quickly. Recall that we maintain the $M(t^{\mathrm{DS}})$ index for each DS (i.e., the total number of tuples that can join with t^{DS}); thus if we find $M(t^{\mathrm{DS}})$ joins with thematic tuples this means that the computed score is actually the final score. Fig. 11 and Fig. 12 illustrate the scalability of HBD and H*k*BD for various |*DS*| and |*Th*| selectivities; runtime trends are similar as for single paths.

In summary, *k*BD and H*k*BD are always faster than their counterpart baseline algorithms; in some cases they are up to two orders of magnitude faster. For instance, for the default setting on the DBLP $\mathcal{JP}_1$, i.e., k = 4 and 20% of |*DS*| and |*Th*|, which corresponds to 68 K and 593 tuples in identifying and thematic relations respectively, *k*BD requires only 0.02 seconds, whereas BD requires 4.3 seconds (i.e., *k*BD is 180 times faster). For the multi-path queries on DBLP $\mathcal{JP}_1$, $\mathcal{JP}_2$ and $\mathcal{JP}_3$ (using again default values with 68 K and 207 K tuples in identifying and thematic relations respectively), H*k*BD requires 4.08 seconds whereas HBD requires 168.8 seconds (i.e., H*k*BD is about 40 times faster).

8. Conclusion

In this paper, we investigate the effective and efficient thematic ranking of OSs. The proposed ranking paradigm comprises (1) an identifying keyword and (2) a thematic set of keywords. Our ranking gracefully combines IR style properties, authoritative ranking and affinity. The proposed paradigm introduces computational challenges, as the proposed ranking formula considers many parameters in a non-monotonic function and involves expensive group by joins. We conducted a thorough investigation and proposed an efficient top-k group-by join algorithm. Our experimental evaluation on DBLP and TPC-H databases verifies both the effectiveness of thematic OS ranking and the efficiency of our proposed algorithm.

In [12,13], the efficient and effective retrieval of size-l OSs (i.e. snippets of OS) was investigated. Our plans for future work include the combination of thematic and other ranking approaches of OSs with the size-l OSs retrieval; namely, we plan to investigate the concurrent generation and ranking of size-l OSs. In addition, we also plan to investigate the diversification of thematic ranking results [30–32].

Acknowledgements

The work of Zhi Cai was supported by the Beijing Natural Science Foundation under grant number 4172004, Beijing Municipal

Education Commission Science and Technology Program under grant number KM201610005022. Nikos Mamoulis was supported by the EU H2020 research and innovation programme under grant agreement number 657347.

References

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, S. Sudarshan, Banks: Browsing and keyword searching in relational databases, in: VLDB, 2002, pp. 1083–1086.
[2] V. Hristidis, Y. Papakonstantinou, Discover: Keyword search in relational databases, in: VLDB, 2002, pp. 670–681.
[3] G. J. Fakas, Automated generation of object summaries from relational databases: A novel keyword searching paradigm, in: DBRank '07, ICDE, 2008, pp. 564–567.
[4] G.J. Fakas, Z. Cai, Ranking of object summaries, in: DBRank '08, ICDE, 2009, pp. 1580–1583.
[5] G.J. Fakas, A novel keyword search paradigm in relational databases: object summaries, Data Knowl. Eng. 70 (2) (2011) 208–229.
[6] V. Hristidis, L. Gravano, Y. Papakonstantinou, Efficient ir-style keyword search over relational databases, in: VLDB, 2003, pp. 850–861.
[7] M. Sydow, M. Pikula, R. Schenkel, The notion of diversity in graphical entity summarisation on semantic knowledge graphs, J. Intell. Inf. Syst. 10 (2) (2013) 1–41.
[8] G. Cheng, T. Tran, Y. Qu, Relin: relatedness and informativeness-based centrality for entity summarization, Semantic Web-ISWC 2011 (2011) 114–129.
[9] A. Dass, C. Aksoy, A. Dimitriou, D. Theodoratos, Exploiting semantic result clustering to support keyword search on linked data, WISE 10 (2) (2014) 448–463.
[10] Y. Luo, X. Lin, W. Wang, X. Zhou, Spark: top-k keyword query in relational databases, in: SIGMOD Conference, 2007, pp. 115–126.
[11] Y. Luo, W. Wang, X. Lin, X. Zhou, J. Wang, K. Li, SPARK2: top-k keyword query in relational databases, IEEE Trans. Knowl. Data Eng. 23 (12) (2011) 1763–1780.
[12] G.J. Fakas, Z. Cai, N. Mamoulis, Versatile size-l object summaries for relational keyword search, IEEE Trans. Knowl. Data Eng. 26 (4) (2014) 1026–1038.
[13] G.J. Fakas, Z. Cai, N. Mamoulis, Diverse and proportional size-l object summaries for keyword search, in: SIGMOD, 2015, pp. 563–574.
[14] G.J. Fakas, Z. Cai, N. Mamoulis, Diverse and proportional size-l object summaries using pairwise relevance, VLDB Journal, 2016.
[15] R. Goldman, N. Shivakumar, S. Venkatasubramanian, H. Garcia-Molina, Proximity search in databases, in: VLDB, 1998, pp. 26–37.
[16] A. Simitsis, G. Koutrika, Y. E. Ioannidis, Generalized précis queries for logical database subset creation, in: ICDE, 2007, pp. 1382–1386.
[17] A. Simitsis, G. Koutrika, Y.E. Ioannidis, Précis: from unstructured keywords as queries to structured databases as answers, VLDB J. 17 (1) (2008) 117–149.
[18] A. Dimitriou, D. Theodoratos, Efficient keyword search on large tree structured datasets, KEYS 10 (2) (2012) 63–74.
[19] A. Dimitriou, D. Theodoratos, T. Sellis, Top-k-size keyword search on tree structured data, Inf. Syst. 10 (2) (2015) (178–19).
[20] S. Bergamaschi, F. Guerra, M. Interlandi, R. Trillo-Lado, Y. Velegrakis, Combining user and database perspective for solving keyword queries over relational databases, Inf. Syst. 10 (2) (2016) (178–19).
[21] R. Fagin, A. Lotem, M. Naor, Optimal aggregation algorithms for middleware, in: PODS, 2001.
[22] U. Güntzer, W.-T. Balke, W. Kießling, Towards efficient multi-feature queries in heterogeneous environments, in: ITCC, 2001, pp. 622–628.
[23] Z. Zhang, S. won Hwang, K. C.-C. Chang, M. Wang, C. A. Lang, Y.-C. Chang, Boolean + ranking: querying a database by k-constrained optimization, in: SIGMOD Conference, 2006, pp. 359–370.
[24] I.F. Ilyas, W.G. Aref, A.K. Elmagarmid, Supporting top-k join queries in relational databases, in: VLDB, 2003, pp. 754–765.
[25] C. Li, K. C.-C. Chang, I.F. Ilyas, Supporting ad-hoc ranking aggregates, in: SIGMOD Conference, 2006, pp. 61–72.
[26] A. Balmin, V. Hristidis, Y. Papakonstantinou, Objectrank: Authority-based keyword search in databases, in: VLDB, 2004, pp. 564–575.
[27] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426.
[28] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, H. Karambelkar, Bidirectional expansion for keyword search on graph databases, in: VLDB, 2005, pp. 505–516.
[29] G.J. Fakas, Z. Cai, N. Mamoulis, Size-l object summaries for relational keyword search, PVLDB 5 (3) (2011) 229–240.
[30] S. Cheng, A. Arvanitis, M. Chrobak, V. Hristidis, Multi-query diversification in microblogging posts, in: EDBT, 2014, pp. 133–144.
[31] A. Kashyap, V. Hristidis, Logrank: Summarizing social activity logs, in: WebDB, 2012, pp. 1–6.
[32] L. Qin, J.X. Yu, L. Chang, Diversifying top-k results, PVLDB 5 (11) (2012) 1124–1135.