Efficient and Effective Aggregate Keyword Search on Relational Databases*

Luping Li[¶] Stephen Petschulat[‡] Guanting Tang[†] Jian Pei[†] Wo-Shun Luk[†]
[¶] Baidu Inc., Beijing, China, liluping01@baidu.com
[‡] SAP Research, Vancouver, BC, Canada, stephen.petschulat@sap.com
[†] Simon Fraser University, Burnaby, BC, Canada, {gta9,jpei,woshun}@cs.sfu.ca

Abstract

Keyword search on relational databases is useful and popular for many users without technical background. Recently, aggregate keyword search on relational databases was proposed and has attracted interest. However, two important problems still remain. First, aggregate keyword search can be very costly on large relational databases, partly due to the lack of efficient indexes. Second, finding the top-k answers to an aggregate keyword query has not been addressed systematically, including both the ranking model and the efficient evaluation methods. In this paper, we tackle the above two problems to improve the efficiency and effectiveness of aggregate keyword search on large relational databases. We design indexes efficient in both size and in construction time. We propose a general ranking model and an efficient ranking algorithm. We also report a systematic performance evaluation using real data sets.

*The work was done when the first author was a master's student at Simon Fraser University. This research is supported in part by an NSERC Discovery Grant, a BCFRST NRAS Endowment Research Team Program project, two SAP Business Objects ARC Fellowships, and two NSERC CRD Research Grants. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

1 Introduction

More and more relational databases contain textual data and thus keyword search on relational databases becomes popular. Aggregate keyword search [22] was recently proposed on relational databases: given a set of keywords, find a set of aggregates such that each aggregate is a group-by covering all query keywords.

Aggregate keyword search on relational databases has attracted a lot of attention [22, 7, 21, 6, 14, 5, 15]. A few critical challenges have been identified, such as how to develop efficient approaches for finding all minimal group-bys [22] or top-k relevant cells [7, 6] to a user given keyword query. To motivate, we revisit the example in [22].

Example 1 (Motivation [22])**.** *Table 1 shows a database of tourism event calendar. Such an event calendar is popular in many tourism web sites and travel agents' databases (or data warehouses). To keep our discussion simple, in the field of* `description`*, a set of keywords are extracted. In general, this field can store text description of events.*

Month	State	City	Event	Description
December	Texas	Houston	Space Shuttle Experience	rocket, supersonic, jet
December	Texas	Dallas	Cowboy's Dream Run	motorcycle, culture, beer
December	Texas	Austin	SPAM Museum Party	classical American Hormel foods
November	Arizona	Phoenix	Cowboy Culture Show	rock music

Table 1: A table of tourism events.

Scott, a customer planning his vacation, is interested in seeing space shuttles, riding motorcycle and experiencing American food. He can search the event calendar using the set of keywords {*"space shuttle", "motorcycle", "American food"*}*. Unfortunately, the three keywords do not appear together in any single tuple, and thus the results returned by the existing keyword search methods may contain at most one keyword in a tuple.*

However, Scott may find the aggregate group (December, Texas, $*$, $*$, $*$) interesting and useful, since he can have space shuttles, motorcycle, and American food all together if he visits Texas in December. The $*$ signs on attributes `city`*,* `event`*, and* `description` *mean that he will have multiple events in multiple cities with different description.*

To make his vacation planning effective, Scott may want to have the aggregate as specific as possible – it should cover a small area (for example, Texas instead of the whole United States) and a short period (for example, December instead of year 2009).

In summary, the task of keyword search for Scott is to find minimal aggregates in the event calendar database such that for each of such aggregates, all keywords are contained by the union of the tuples in the aggregate. ■

Two problems still remain for aggregate keyword search. First, aggregate keyword search is still costly on large relational databases, partly due to the lack of efficient indexes. For example, the keyword graph index [22] is used to generate all aggregate groups for a keyword query. However, it

Dataset	ConstructionTime	Space Consumption
e-Fashion (308KB)	$2hour57mins$	$\geq 1.0GB$
SuperstoreSales (2MB)	$> 3hour$	$\geq 1.5GB$
CountryInfo (19KB)	$17mins$	$\geq 0.5GB$

Table 2: The construction time and space consumption of the keyword graph index [22] for some real datasets.

often takes a long time to construct the index on large database and has a large space consumption, as demonstrated in Table 2 using some real data sets to be discussed in detail in Section 5.

The second problem is that finding the top-k answers to an aggregate keyword query has not been addressed systematically. Since aggregate keyword search on large relational databases may find a large number of answers, ranking the answers effectively becomes important. It is necessary to develop efficient top-k algorithm to find the top-k most relevant aggregates. Although [7, 6] develop efficient methods to find top-k relevant cells for an aggregate keyword query, such a relevant cell may not match all the query keywords. [22] proposes two approaches to find all the minimal group-bys for an aggregate keyword query and each minimal group-by matches all the query keywords, but these minimal group-bys are not ranked and there is no top-k algorithm in [22].

In this paper, we tackle the above two problems to improve the efficiency and effectiveness of aggregate keyword search on large relational databases. We design indexes efficient in both size and construction time. We propose a general ranking model and an efficient ranking algorithm. We also report a systematic performance evaluation using real data sets.

The rest of the paper is organized as follows. In Section 2, we formulate the aggregate keyword search problem and review the previous studies related to our work. We discuss the index design in Section 3. The top-k query answering method is presented in Section 4. We report an empirical evaluation in Section 5, and finally conclude the paper in Section 6.

2 Problem Definition and Related Work

We follow the terminology in [22] throughout the paper. We revisit the preliminaries and state the problem in Section 2.1. We review the related works in Section 2.2.

2.1 Preliminaries and Problem Definition

Let $T = (A_1, \ldots, A_n)$ be a relational table. A **group-by** on table T is a tuple $c = (x_1, \ldots, x_n)$ where $x_i \in A_i$ or $x_i = *$ $(1 \leq i \leq n)$, and $*$ is a meta symbol meaning that the attribute is generalized. The **cover** of group-by c is the set of tuples in T that have the same values as c on those non-$*$ attributes, that is, $\mathbf{Cov(c)} = \{(v_1, \ldots, v_n) \in T | v_i = x_i \text{ if } x_i \neq *, 1 \leq i \leq n\}$.

A **base group-by** is a group-by which takes a non-$*$ value on every attribute. For two group-bys $c_1 = (x_1, \ldots, x_n)$ and $c_2 = (y_1, \ldots, y_n)$, c_1 is an **ancestor** of c_2, and c_2 a **descendant** of c_1, denoted by $c_1 \succ c_2$, if $x_i = y_i$ for each $x_i \neq *(1 \leq i \leq n)$, and there exists $k(1 \leq k \leq n)$ such that

$x_k = *$ but $y_k \neq *$.

Given a table T , an **aggregate keyword query** is a 3-tuple $q = (D, C, W)$, where D is a subset of attributes in table T , C is a subset of text-rich attributes in T , and W is a set of keywords. We call D the **aggregate space** and each attribute $A \in D$ a **dimension**. We call C the set of **text attributes** of q. D and C do not have to be exclusive to each other.

A group-by c is a **minimal answer** to an aggregate keyword query q if c is an answer to q and every descendant of c is not an answer to q. As mentioned in Section 1, users may prefer specific information, so our method needs to guarantee that every returned group-by is minimal.

For a set of tuples t_1 and t_2 in table T, the **max-join** of t_1 and t_2 is a tuple $t =$ "$t_1 \vee t_2$" such that for any attribute A in T, $t[A] = t_1[A]$ if $t_1[A] = t_2[A]$, otherwise $t[A] = *$. We call $(*, *, \ldots, *)$ a **trivial answer**.

Theorem 1 (Max-join on answers [22])**.** *If t is a minimal answer to aggregate keyword query $q = (D, C, \{w_1, \cdots, w_m\})$, then there exists minimal answers t_1 and t_2 to queries $(D, C, \{w_1, w_2\})$ and $(D, C, \{w_e, \cdots, w_m\})$, respectively, such that $t = t_1 \vee t_2$.*

To answer query $q = (D, C, \{w_1, \ldots, w_m\})$, using Theorem 1 repeatedly, we only need to check $m - 1$ edges covering all keywords $w_1, \ldots, w_m$ in the clique. Each edge is associated with the set of minimal answers to a query on a pair of keywords. The weight of the edge is the size of the answer set. In order to reduce the total cost of the joins, heuristically, we can find a spanning tree connecting the m keywords such that the product of the weights on the edges is minimized.

Given a table T, a **keyword graph index** is an undirected graph $G(T) = (V, E)$ such that V is the set of keywords in the table T and $(u, v) \in E$ is an edge, if there exists a non-trivial answer to query $q_{uv} = (D, C, \{u, v\})$. Edge (u, v) is associated with the set of minimal answers to query q_{uv}. Zhou and Pei [22] proved that, if there exists a nontrivial answer to an aggregate keyword query q, the keyword graph index exists a clique on all keywords of q (Theorem 3 in [22]).

We define a **query keyword graph** as follows.

Definition 1 (Query keyword graph)**.** *Given a table T , a* ***query keyword graph*** *for an aggregate keyword query $q = (D, C, \{w_1, \cdots, w_m\})$ is an undirected graph $G(T, Q) = (V, E)$ such that $V = \{w_1, \cdots, w_m\}$ is the set of query keywords and $(w_i, w_j) \in E$ is an edge if there exists a non-trivial answer to query $(D, C, \{w_i, w_j\})$. Edge (w_i, w_j) is associated with the set of minimal answers to query $(D, C, \{w_i, w_j\})$, $1 \leq i, j \leq m$.* ■

Example 2 (Keyword graph index and query keyword graph)**.** *In Table 3, a table T has 3 text attributes and 3 tuples (or base group-bys). The keywords are w_1, w_2, w_3 and w_4. We perform max-join on each pair of tuples in table T and get the following group-bys: $g_1 : (*, w_3, w_2)$, $g_2 : (w_1, w_3, *)$, $g_3 : (*, w_3, *)$, $r_1 : (w_1, w_3, w_2)$, $r_2 : (w_4, w_3, w_2)$, and $r_3 : (w_1, w_3, w_4)$. Among them, r_1, r_2 and r_3 are base group-bys.*

The corresponding keyword graph index is shown in Figure 1. Each edge (w_i, w_j) in Figure 1 contains a set of group-bys and each such a group-by is a minimal answer to the query $(D, C, \{w_i, w_j\})$. For example, edge (w_1, w_2) contains a base group-by r_1, which is a minimal answer to the query $(D, C, \{w_1, w_2\})$.

RowID	$TextAttri_1$	$TextAttri_2$	$TextAttri_3$
r_1	w_1	w_3	w_2
r_2	w_4	w_3	w_2
r_3	w_1	w_3	w_4

Table 3: An example of table T

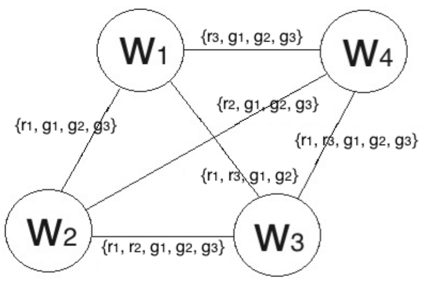


Figure 1: A keyword graph index

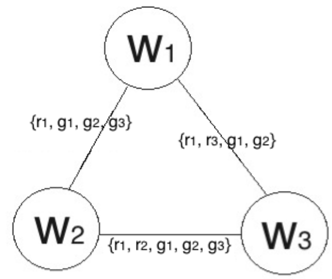


Figure 2: A query keyword graph

For the aggregate keyword query $(D, C, \{w_1, w_2, w_3\})$*, a corresponding query keyword graph, as shown in Figure 2, can be constructed.* ∎

Obviously, for an aggregate keyword query q , there exists a non-trivial answer to q in table T if and only if in the query keyword graph $G(T, q)$ is a clique.

The number of edges in a keyword graph index is $O(|V|^2)$, where V is the set of keywords in the relational database. For a small relational database, the number of keywords in the database is limited and the corresponding keyword graph index can be maintained easily. As the database grows larger, the number of keyword increases and the keyword graph index becomes less efficient.

The difference between a query keyword graph and a keyword graph index is that vertices of the former are keywords in the query q and vertices of the latter are keywords in the whole database. Since the number of keywords in a query is much smaller than that in a relational database, a query keyword graph is often much smaller than a keyword graph index and can be constructed quickly.

Our first task in this paper is to build a new index to facilitate constructing a query keyword graph during the query processing period. The aggregate information in the query keyword graph is then used to generate minimal answers. Our complete query-answering method successfully uses the new index to generate all the minimal answers to a keyword query.

Although many non-minimal answers are pruned during the query processing period, the number of minimal answers to a keyword query may still be large. For example, as we will discuss in detail in Section 5, there are about $1,000$ minimal answers to some aggregate keyword queries on the SuperstoreSales dataset, which has $8,300$ tuples and 21 dimensions. To tackle the problem, we investigate finding top-k minimal answers. We define several features on the group-bys. The overall score function of the group-by is a linear combination of those features. We also develop efficient pruning methods to quickly find the top-k results.

2.2 Related Work

In general, our study is related to the existing work on keyword search on relational databases and keyword-based search in data cube. In this section, we review some representative studies and point out the differences between those studies and our work.

2.2.1 Keyword Search on Relational Databases

Keyword search on relational databases is an active topic in database research nowadays. Zhou and Pei [22] studied keyword based aggregation on large relational databases using minimal group-bys, which is most related to our study.

Given a table, Zhou and Pei [22] constructed a keyword graph index, which is used during the online query processing phase to generate all minimal answers that contain all the user given keywords. Each edge in the keyword graph index is corresponding to a pair of keywords. Minimal answers to every pair of keywords are pre-calculated and stored in the keyword graph index. To answer an aggregate keyword query q, their method first scans the keyword graph index to check if there exists a clique on all the query keywords. If so, it then performs max-join repeatedly on $|Q| - 1$ edges in that clique and finds nontrivial minimal answers from the max-join results. If not, there are no nontrivial minimal answers to q.

In this paper, we will design a new index which is more efficient than the keyword graph index [22]. The new index can be used to quickly construct small query keyword graphs serving the same purpose as a keyword graph index. We also develop efficient and effective methods to rank the minimal answers.

There are also a number of works on relational databases in the literature. For example, Balmin *et al.* [2] treated the database as a labeled graph and built a labeled graph index which has a natural flow of authority. Given a keyword query, they applied a PageRank algorithm to find nodes in the labels graph that have high authority with respect to all query keywords. Hristidis *et al.* [12] built an index combining a set of joining networks, each representing a row that can be generated by joining rows in multiple tables using primary and foreign keys. Given a keyword query, they scaned the index to find relevant joining networks each containing all the query keywords. Agrawal *et al.* [1] implemented a keyword-based search system DBXplorer on a commercial database. It returns the relevant rows as answers such that each relevant row contains all the query keywords. Its index contains a symbol table that can help to quickly locate the query keywords in the relational database. Bhalotia *et al.* [4] designed a graph index on relational database. Each node represents a row and each edge represents an application-oriented relationship between two rows. Given a keyword query, their method scans the index to find Steiner trees [11] that contain all the query keywords.

All the above studies [2, 12, 1, 4] focus on finding relevant tuples instead of aggregate cells, so their indexes, score functions and top-k algorithms can not be extended to solve our problems directly.

2.2.2 Keyword-based Search in Data Cubes

Following the framework of [22], Ding *et al.* [7, 6] found the top-k most relevant cells for a keyword query on a data cube with text-rich dimensions. A base group-by is treated as a document and the documents covered by a cell C_{cell} is treated as a "big document" (also called the cell document of C_{cell}, represented by $C_{cell}[D_{cell}]$). The relevance score of a cell C_{cell} is defined as a function $rel(q, C_{cell})$ of the cell document $C_{cell}[D_{cell}]$ and the query q. They use an IR style model to design the score function of a cell [7] .

$$rel(q, C_{cell}) = \sum_{t \in q} \ln \frac{N - df_t + 0.5}{df_t + 0.5} \times \frac{(k_1 + 1)tf_{t,D_{cell}}}{k_1((1-b) + b\frac{dl_D}{avdl}) + tf_{t,D_{cell}}} \times \frac{(k_3 + 1)qtf_{t,q}}{k_3 + qtf_{t,q}}$$

where N is the number of rows in the database, D_{cell} is the big document of C_{cell}, $tf_{t,D_{cell}}$ is the term frequency of term $t \in q$ in D_{cell} , df_t is the number of documents in the database containing t, dl_D represents the length of D_{cell}, $avdl$ is the average length of documents covered by C_{cell}, $qtf_{t,q}$ is the number of times t appearing in q, and k_1,b, k_3 are the parameters used in Okapi BM25 [17, 16].

Since the parameters of Okapi BM25 are query and collection (cell) dependent, this score function is sensitive to parameters.

To find the top-k relevant cells, Ding *et al.* [7] proposed four approaches: inverted-index one-scan, document sorted-scan, bottom-up dynamic programming, and search-space ordering. Ding *et al.* [6] proposed another two approaches: TACell and BoundS.

The inverted-index one-scan method generates and scores all the non-empty cells. Since the number of non-empty cells increases exponentially with respect to the dimensionality of the database, this method is efficient only when the number of dimensions is small (from 2 to 4). The document sorted-scan approach uses a priority queue to keep candidate cells in the relevance descending order. All rows (documents) of the database are scanned in the relevance descending at the beginning. Similar to the inverted-index one-scan method, once a row is scanned, all the cells covering it are explored. It then calculates the relevance scores of the explored cells. Finally, if an explored cell does not cover any non-scanned rows in the database and the number of its covered rows is larger than a threshold, it would be inserted into the priority queue. Top-k cells are selected from the priority queue. For this method, once a row is scanned, 2^n cells are explored in an n-dimension cube. So the numbers of candidate cells and explored cells increase very quickly. Although the complexity of this method is worse than the inverted-index one-scan, it may terminate earlier before scanning all rows.

Different from the above one-scan and sorted-scan approaches that compute the relevance score of a cell from rows in the database, the bottom-up approach and the search-space ordering approach compute the score of a cell from its children cells in a dynamic-programming manner. Since the score of a cell on a certain level can be quickly calculated from its children cells on the lower level, which is faster than computing from cells on the base level, the bottom-up is more efficient than the previous two approaches. However, the bottom-up method still needs to calculate the scores of all the cells, so it is efficient only when the number of dimensions is small.

The search-space ordering method carries out cell-based search and explores an as small as possible number of cells in the cube to find the top-k answers. With some pruning techniques, this method avoids exploring all cells in the text cube and is more efficient than the previous three approaches.

The above four approaches do not pre-process the database to build any index offline.Ding *et al.* [6] developed another two approaches, TACell and BoundS, which build indexes offline.

The TACell method extends the threshold algorithm (TA) [9] for finding the top-k relevant cells with respect to a given keyword query q. It treats each cell as a ranking object in TA and needs to build an offline index containing many sorted lists. Given a database, it first generates all non-empty cells; for each term t in the database, it creates a sorted list of cells L_t, where the generated cells are sorted in the descending order of term frequency of t in each cell document (big document). It also creates another sorted list L_{len}, where cells are sorted in the ascending order of the lengths of cell documents. So, if the n-dimension database (N rows) contains M terms, the number of sorted lists is $M + 1$. On large relational databases, the number of terms is huge and the total number of non-empty cells is $\Omega(N * 2^n)$. Such an index may not be efficient since it may be too large to fit into main memory in whole.

The index of BoundS only contains some inverted indices for all terms with respect to the rows in the database. Compared with TACell, BoundS is more efficient in building the offline index but consumes more time for online queries. The basic idea of online processing in BoundS is to estimate and update the lower bounds and upper bounds of the relevance scores of the cells (explored when scanning the database rows) to prune some non-top-k cells.

TACell and BoundS apply an IR-style relevance model for scoring and ranking cell documents in the text cube. For a query $q = \{t_1, t_2, ..., t_l\}$, $rel(q, C_{cell}) = s(tf_{t_1}, tf_{t_2}, ..., tf_{t_l}, |D_{cell}|)$, where tf_{t_i} is the term frequency (the occurrence count of a term in a document [18, 19]) of the i_{th} term of q in the cell document D_{cell} of C_{cell}, and s is a user defined function.

The score function $s()$ needs to be monotonic to ensure the correctness of TACell and BoundS. Ding *et al.* [6] used a simple monotonic function that considers term frequencies and document length (terminology in IR). In BoundS, it is assumed that the length of the big document for each cell, i.e., the document length, is precomputed. Thus, only the term frequency is needed when estimating the lower bounds and upper bounds of the relevance scores of the cells. If more IR features (such as df_t and $qtf_{t,q}$) are considered in the score function, more sorted lists need to be created in TACell and thus the index has a larger space consumption. Moreover, the upper bounds and lower bounds defined in BoundS may no longer be applicable.

In addition, Zhao *et al.* [21] and Wu *et al.* [20] supported interactive exploration of data using keyword search. Wu *et al.* [20] proposed a system (KDAP) that supports interactive exploration of data using keyword search. Given a keyword query, the system first generates the candidate subspaces in an OLAP database such that each subspace essentially corresponds to a possible join path between the dimensions and the facts. It then ranks the subspaces and asks users to select one subspace. Finally, it computes the group-by aggregates over some predefined measure using qualified fact points in the selected subspace and finds the top-k group-by attributes to partition the subspace.

B. Zhao *et al.* [21] proposed a similar keyword-based interactive exploration framework called

TEXplorer. Different from the work in [22, 7, 6], whose goal is to return a ranked list of the cells directly, TEXplorer guides users to find their interested information step by step.

More related work can be found in [5, 13], which give an overview of the state-of-the-art techniques for supporting keyword-based search and exploration on databases. Different from our work, the top-k cells found in [7, 6] are not guaranteed to contain all the query terms. Moreover, [21, 20] address a different application scenario from us. In this paper, we extend [22] and focus on the efficiency and the effectiveness issues of aggregate keyword search on relational databases.

3 An Efficient Index

To make aggregate keyword search more efficient on large relational databases, we design a new index, which is smaller and faster to construct. The new index can be used to correctly generate the same minimal aggregates as the keyword graph index [22].

3.1 The Index

Our new index is called Inverted Pair-wise Joins (IPJ), which stores only the necessary information that can be used to quickly generate the same clique as is used in the keyword graph index [22] during the query processing period.

Definition 2. *Given a table T, the IPJ index stores*

the pair-wise joins of a keyword w**.** $PJ[w] = \{gb|gb$ *is a group-by such that* $gb = r_i \vee r_j$*, where* w *is a keyword in* T*,* (r_i, r_j) *is a pair of rows in* T*, and* $w \in r_i$ *or* $w \in r_j\}$*; and*

the inverted pair-wise joins. $IPJ = \{(w, PJ[w])|w$ *is a keyword in the table* $T\}$.

For each keyword w in the table, the inverted pair-wise joins IPJ records the corresponding pair-wise joins of w ($PJ[w]$). $PJ[w]$ stores without redundancy all relevant group-bys (non-trivial) such that each relevant group-by is generated by performing max-join operation on a certain pair of rows (at least one row contains the keyword w).

Example 3 (The Inverted Pair-wise Joins)**.** *In Table 4, a table T has $m = 4$ text attributes, $n = 4$ rows (r_1, r_2, r_3 and r_4), and $p = 12$ different keywords. Each dimension has $p' = 3$ different values. Since a group-by may take value $*$ on a dimension, there are $(p' + 1)^m = (3 + 1)^4 = 256$ possible group-bys and 255 of them are non-trivial group-bys. The index of TACell [6] needs to store $(p + 1) \times 255 = 3315$ group-bys. For the keyword graph index, there are $\frac{p\times(p-1)}{2} = 66$ edges inside. If the average number of minimal answers on an edge is 2, the keyword graph index needs to store $66 \times 2 = 132$ group-bys. How many group-bys does IPJ need to store?*

We first perform max-join on each pair of rows in table T and get the following group-bys:

- *base group-bys* $r_1 : (w_{11}, w_{21}, w_{31}, w_{41})$, $r_2 : (w_{11}, w_{22}, w_{32}, w_{42})$, $r_3 : (w_{12}, w_{22}, w_{33}, w_{43})$, *and* $r_4 : (w_{13}, w_{23}, w_{33}, w_{41})$*; and*

RowID	$TextAttri_1$	$TextAttri_2$	$TextAttri_3$	$TextAttri_4$
r_1	w_{11}	w_{21}	w_{31}	w_{41}
r_2	w_{11}	w_{22}	w_{32}	w_{42}
r_3	w_{12}	w_{22}	w_{33}	w_{43}
r_4	w_{13}	w_{23}	w_{33}	w_{41}

Table 4: A table T

$Keywords$	PJ[w]
w_{11}	r_1, r_2, g_1, g_3, g_4
w_{12}	r_3, g_4, g_6
w_{13}	r_4, g_3, g_6
w_{21}	r_1, g_1, g_3
w_{22}	r_2, r_3, g_1, g_4, g_6
w_{23}	r_4, g_3, g_6
w_{31}	r_1, g_1, g_3
w_{32}	r_2, g_1, g_4
w_{33}	r_3, r_4, g_3, g_4, g_6
w_{41}	r_1, r_4, g_1, g_3, g_6
w_{42}	r_2, g_1, g_4
w_{43}	r_3, g_4, g_6

Table 5: IPJ of table T

- *aggregate group-bys* $g_1 : (w_{11}, *, *, *) = r_1 \vee r_2$, $g_2 : (*, *, *, *) = r_1 \vee r_3$, $g_3 : (*, *, *, w_{41}) = r_1 \vee r_4$, $g_4 : (*, w_{22}, *, *) = r_2 \vee r_3$, $g_5 : (*, *, *, *) = r_2 \vee r_4$, *and* $g_6 : (*, *, w_{33}, *) = r_3 \vee r_4$.

The trivial group-bys g_2 *and* g_5 *are pruned. The inverted pair-wise joins IPJ (Table 5) can then be generated according to its definition. For example, we know that only the row* r_3 *contains the keyword* w_{12}*. To generate the pair-wise joins for* w_{12}*, we only need to perform max-join operations on* $(r_3, r_3), (r_3, r_1), (r_3, r_2)$ *and* (r_3, r_4)*, the corresponding max-join results are* r_3*,* g_2*,* g_4 *and* g_6*. Since Group-by* g_2 *is trivial and should be pruned,* $PJ[w_{12}] = \{r_3, g_4, g_6\}$*.*

Our inverted pair-wise joins IPJ needs to store only 44 group-bys. ■

To further reduce the size of our new index, we can prune duplicate group-bys by storing all generated group-bys in a set and replace each group-by in the inverted pair-wise joins with its unique identity in this set.

3.2 Using IPJ in Query Answering

To answer a query $q = (D, C, \{w_1, \ldots, w_h\})$, the complete query-answering method first constructs a query keyword graph using our new IPJ index.

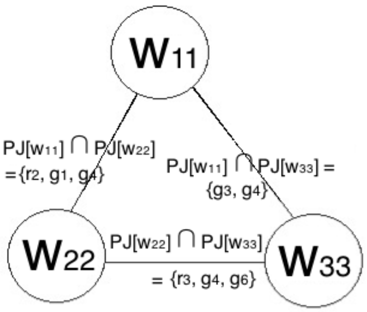


Figure 3: A query keyword graph

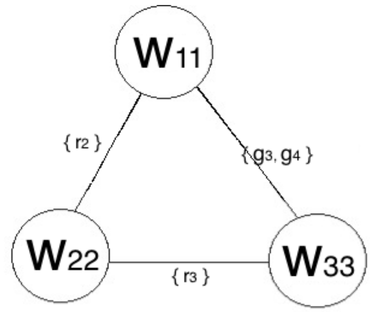


Figure 4: A query keyword graph after pruning non-minimal answers

Example 4 (Query answering)**.** *Consider query $q = (D, C, \{w_{11}, w_{22}, w_{33}\})$ on table T in Table 4. A query keyword graph as shown in Figure 3 is then quickly constructed. The graph is a clique and each node of the graph is a query keyword in the query. For each edge (w_i, w_j) in the clique, the corresponding candidate answers are the intersection of $PJ[w_i]$ and $PJ[w_j]$ in the IPJ index (Table 5). After pruning non-minimal answers on each edge, the query keyword graph is as shown in Figure 4.* ■

To answer query $q = (D, C, \{w_1, \ldots, w_m\})$, using Theorem 1 repeatedly, we only need to check $m - 1$ edges covering all keywords $w_1, \ldots, w_m$ in the clique. Each edge is associated with the set of minimal answers to a query on a pair of keywords. In the above example, the query contains 3 keywords, so only 2 edges need to be checked. The weight of an edge is the size of the corresponding answer set. In order to reduce the total cost of the joins, heuristically, we can find a spanning tree connecting the m keywords such that the product of the weights on the edges is minimized.

If t is a minimal answer to aggregate keyword query $q = (D, C, \{w_1, \cdots, w_m\})$, then there exists minimal answers t_1 and t_2 to queries $(D, C, \{w_1, w_2\})$ and $(D, C, \{w_e, \cdots, w_m\})$, respectively, such that $t = t_1 \vee t_2$.

Example 5 (Checking edges)**.** *Continued from Example 4,*

- *If we check edge (w_{11}, w_{22}) and edge (w_{22}, w_{33}), to generate the candidate answers, we need to perform max-join operations on (r_2, r_3). The corresponding result is group-by g_4.*
- *If we check edge (w_{11}, w_{22}) and edge (w_{11}, w_{33}), to generate the candidate answers, we need to perform max-join operations on (r_2, g_3) and (r_2, g_4). The corresponding results are a trivial group-by and group-by g_4.*
- *If we check edge (w_{22}, w_{33}) and edge (w_{11}, w_{33}), to generate the candidate answers, we need to perform max-join operations on (r_3, g_3) and (r_3, g_4). The corresponding results are a trivial group-by and group-by g_4.*

Thus, no matter which two edges are checked, after pruning unsatisfied (duplicated, trivial, or non-minimal) group-bys, the results are the same. In the above example, the complete query-

Algorithm 1 The new index construction algorithm.

Require:

A table T;

Ensure:

The new index IPJ

Create L_1= { $(w, R[w])$ | w is a keyword in T, $R[w]$ represents all the rows that contain w };
Create L_2= { $(r, S[r])$ | r is a row in T, the corresponding set $S[r] = NULL$ };
for each row $r_1 \in T$ do **do**
  for each row $r_2 \in T$ do **do**
    g= $r_1 \vee r_2$;
    Add g into $S[r_1]$ and add g into $S[r_2]$;
  end for
end for
Create an Inverted Pair-wise Joins IPJ={ $(w, PJ[w])$ | w is a keyword in T, the corresponding Pair-wise Joins $PJ[w] = NULL$ }
for each item $(w, R[w]) \in L_1$ do **do**
  for each row $r \in R[w]$ do **do**
    Move group-bys from $S[r]$ into $PJ[w]$;
    Prune duplicated group-bys in $PJ[w]$;
  end for
end for
return The inverted pair-wise joins IPJ={ $(w, PJ[w])$ | w is a keyword in T, $PJ[w]$ is the corresponding Pair-wise Joins }

answering method finds one minimal answer (group-by g_4) for the query $q = (D, C, \{w_{11}, w_{22}, w_{33}\})$.

■

3.3 The Index Construction Algorithm

To construct the IPJ index on a table T, we first create an inverted index L_1 to record the information about which rows contain a certain keyword. We then conduct a max-join operations on each pair of rows in the table T to construct another inverted index $L_2 = \{(r, S[r]) | r$ is a row in T, the corresponding set $S[r]$ is null at the beginning$\}$. For example, if the group-by g is the max-join result of rows r_1 and r_2, we add g into $S[r_1]$ and $S[r_2]$. Finally, we join L_1 and L_2 to generate our Inverted Pair-wise Joins. The process summarized in Algorithm 1.

The IPJ index has two advantages comparing to the keyword graph index [22].

First, IPJ is smaller and faster to construct. The space complexity of the keyword graph index [22] is $O(m^2 \times n \times p)$, where m is the number of unique keywords in the table, n is the number of dimensions and p is the average number of minimal answers on each edge in the graph. The space complexity of the IPJ index is $O(m \times n \times p'')$, where p'' is the average size of $PJ[w]$ (w is a keyword).

Dataset	NumOfEdges	KeywordGraphIndex	IPJ
e-Fashion	10^7	$2hour57mins$	$20seconds$
SuperstoreSales	10^{11}	$> 3hour$	$6mins$
CountryInfo	10^6	$17mins$	$8seconds$

Table 6: Construction time of the keyword graph index and IPJ

Given a table T with $m = 10^4$ unique keywords and $n = 10$ dimensions, assume that the average number of minimal answers on each edge is $p = 5$, if we use an integer (4 bytes) to represent a dimension value, the size of a minimal answer is $p' = 40$ bytes. The size of the keyword graph index is about 10 GB. Assuming that the average size of $PJ[w]$ (w is a keyword) is $p'' = 100$, the size of IPJ is $p'' \times p' \times m = 100 \times 40 \times 10^4 = 40 \times 10^6$ bytes, which is about 40 MB.

Table 6 shows the construction time of the two indexes on three real data sets (details in Section 5), from which we can see that our new index is more efficient.

Second, the IPJ index is easier to maintain. When a keyword is deleted, to maintain the keyword graph index [22], we need to find all the corresponding edges and then delete them. So, every edge in the keyword graph index [22] must be checked and the time complexity is $O(m^2)$, where m is the number of unique keywords in the table. To maintain our new index, we only need to delete the corresponding item from the inverted pair-wise joins and the time complexity is $O(m)$.

4 A Top-k Query Answering Algorithm

In this section, we propose a general ranking model and an efficient ranking algorithm.

4.1 Scoring Functions

We define three scoring functions on a group-by: the density score, the dedication score and the structure degree. The overall score of a group-by is a linear combination of these three scores. Table 7 presents the symbols and formulae used in this section.

4.1.1 Density Score

We use a density score to measure whether the query keywords appear frequently in the minimal answers. If a group-by has a high density score, it means that query keywords appear frequently in this group-by, and thus this group-by should be ranked high in the search engine.

The feature of term frequency is often used in IR technologies [18, 19]. Since each group-by covers a set of rows in the table T, we can treat these covered rows as a document and similarly consider the query term frequency in these covered rows.

Definition 3 (Density Score)). *Given an aggregate query Q, the* ***density score*** *of a group-by g is defined as*

Item	Symbol
The threshold on the overall score of k generated answers	s
An aggregate keyword query	$Q, Q = (D, C, \{w_1, \ldots, w_n\})$
The number of query terms in Q	$\|Q\|$
Query terms	$w_i, 1 \leq i \leq n$
A table of the relational database	T
One minimal answer	g
One black node (group-by) on an edge of the query keyword graph	A_i
The set of rows covered by g	$Cov(g), Cov(g) = r_1, \ldots, r_m$
In $Cov(g)$, the number of rows that contain w_i	$N_i, 1 \leq i \leq n$
The set of sub-queries of Q	$C, C = c_1, \ldots, c_y$
One sub-query of Q	$c_j, 1 \leq j \leq y$
In $Cov(g)$, the number of rows that contain c_j	$M_j, 1 \leq j \leq y$
The occurrences of query terms in g	$Num(Q, g)$
The total number of keywords in g	$Num(g)$
The density score of g	$Density(g) = \frac{Num(Q,g)}{Num(g)}$
In T, the number of rows that contain w_i	$DF(w_i), IDF(w_i) = \frac{1}{DF(w_i)}, 1 \leq i \leq n$
The dedication score of g	$Dedication(g) = \sum_{i=1}^{n} IDF(w_i) \times \frac{N_i}{\|Cov(g)\|}$
The structure degree of g	$StructureDegree(g) = \sum_{j=1}^{y} \frac{\|c_j\|}{\|Q\|} \times \frac{M_j}{\|Cov(g)\|}$

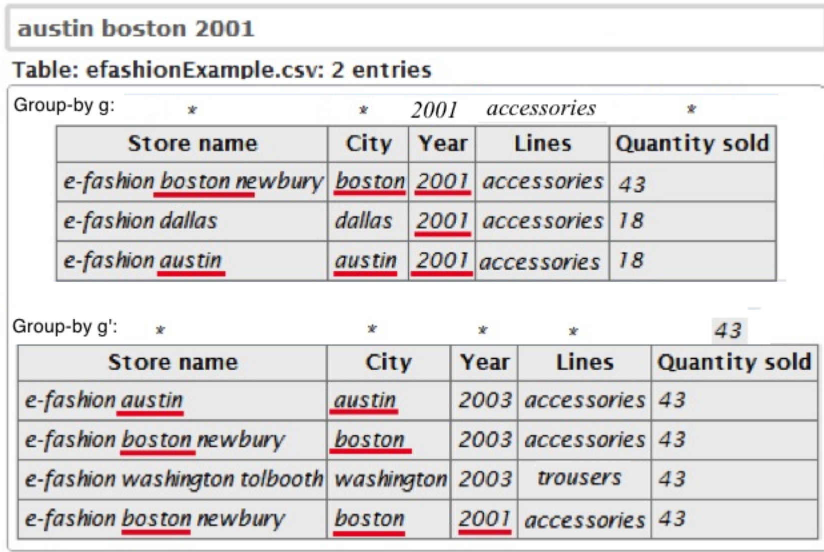
Table 7: Symbols and formulae used in Section 4

$$Density(g) = Density(Cov(g)) = \frac{Num(Q, g)}{Num(g)} \tag{1}$$

where $Num(Q, g)$ is the total number of occurrences of query terms in the group-by g, $Num(g)$ represents the total number of keywords in g, and $Cov(g)$ represents rows covered by g. ■

We calculate the density score of a group-by g using the information in its covered rows ($Cov(g)$). Therefore, $Density(g)$ and $Density(Cov(g))$ are the same.

Example 6 (Density Score)**.** *In Figure 5, suppose the aggregate keyword search engine returns two minimal group-bys for a query $q = (D, C, \{Austin, Boston, 2001\})$. For simplicity, we assume all the attributes are text attributes unless otherwise specified. The two results are $g = (*, *, 2001, accessories, *)$ and $g' = (*, *, *, *, 43)$. The number of keywords in group-by g is $Num(g) = 19$, and the number of query terms in g is $Num(q, g) = 7$. So, the density score of*



austin boston 2001

Table: efashionExample.csv: 2 entries

Group-by g: * * 2001 accessories *

Store name	City	Year	Lines	Quantity sold
e-fashion boston newbury	boston	2001	accessories	43
e-fashion dallas	dallas	2001	accessories	18
e-fashion austin	austin	2001	accessories	18

Group-by g': * * * * 43

Store name	City	Year	Lines	Quantity sold
e-fashion austin	austin	2003	accessories	43
e-fashion boston newbury	boston	2003	accessories	43
e-fashion washington tolbooth	washington	2003	trousers	43
e-fashion boston newbury	boston	2001	accessories	43

Figure 5: A query-answering example

group-by g is $Density(g) = \frac{7}{19} = 0.37$. *Similarly, the number of keywords in* g' *is* $Num(g') = 28$, *and the number of query terms in* g' *is* $Num(q, g') = 7$, *so the density score of group-by* g' *is* $Density(g') = \frac{7}{28} = 0.25$. ■

4.1.2 Dedication Score

Inverted document frequencies (IDF) are often used IR technologies [18, 19], too. Carrying the same spirit, we use a dedication score to measure whether terms with high IDF scores appear frequently in the minimal answers. If a group-by has a high dedication score, it means that some terms with high IDF scores appear frequently in this group-by, and thus this group-by should be ranked high in the search engine.

In a text-rich relational database, some terms may appear in many rows while others may only appear in few rows, if we treat a row as a document, we can similarly consider the IDF feature of a group-by.

Definition 4 (Dedication Score)**.** *Given a query* $Q = (D, C, \{w_1, \ldots, w_n\})$, *the* ***dedication score*** *of a group-by g is defined as*

$$Dedication(g) = Dedication(Cov(g)) = \sum_{i=1}^{n} IDF(w_i) \times \frac{N_i}{|Cov(g)|} \tag{2}$$

where $IDF(w_i)$ *is the inverted value of* $DF(w_i)$, $DF(w_i)$ *is the number of rows that contain a query term* w_i, *and* N_i *is the number of rows (in* $Cov(g)$ *) contain the term* w_i. *We use* $\frac{N_i}{Cov(g)}$ *to measure the weight of* w_i *in* g. *The group-by gis highly dedicated to the term* w_i *if most of its*

StoreName	City	Year	Lines	QuantitySold
e-Fashion Austin	Austin	2003	accsesories	43
e-Fashion Boston Newbury	Boston	2003	accessories	43
e-Fashion Washington Tolbooth	Washington	2003	trousers	43
e-Fashion Boston Newbury	Boston	2001	accessories	43
e-Fashion Dallas	Dallas	2001	accessories	18
e-Fashion Washington Tolbooth	Washington	2002	trousers	18
e-Fashion Washington Tolbooth	Washington	2003	dresses	18
e-Fashion Austin	Austin	2001	accessories	18

Table 8: Query Keywords in the e-Fashion Database

covered rows contain w_i. We use $IDF(w_i) \times \frac{N_i}{|Cov(g)|}$ to measure how g is dedicated to the term w_i. ∎

The dedication score of a group-by g is calculated using the information in its covered rows ($Cov(g)$). Thus, $Dedication(g)$ and $Dedication(Cov(g))$ are the same.

Example 7 (Dedication Score)**.** *Continued from Example 6, suppose the database is as shown in Table 8, and the query is ("Austin", "Boston","2001"). In the database, the number of rows that contain "Austin" is 2, the number of rows that contain "Boston" is 2, and the number of rows that contain "2001" is 3, so the IDF scores of the query terms are $IDF(\text{"Austin"}) = \frac{1}{2} = 0.5$, $IDF(\text{"Boston"}) = \frac{1}{2} = 0.5$, and $IDF(\text{"2001"}) = \frac{1}{3} = 0.33$.*

In Figure 5, the number of rows covered by group-by $g = (*, *, 2001, accessories, *)$ is $|Cov(g)| = 3$, the number of rows in $Cov(g)$ that contain (Austin) is $N_1 = 1$, the number of rows in $Cov(g)$ that contain "Boston" is $N_2 = 1$ and the number of rows in $Cov(g)$ that contain "2001" is $N_3 = 3$. So, the dedication score of group-by g is $Dedication(g) = 0.5 \times \frac{1}{3} + 0.5 \times \frac{1}{3} + 0.33 \times \frac{3}{3} = 0.66$. Similarly, the number of rows covered by group-by $g' = (*, *, *, *, 43)$ is $|Cov(g')| = 4$, the number of rows in $Cov(g')$ that contain "Austin" is $N_1 = 1$, the number of rows in $Cov(g')$ that contain "Boston" is $N_2 = 2$ and the number of rows in $Cov(g')$ that contain "2001" is $N_3 = 1$. So, the dedication score of group-by g' is $Dedication(g') = 0.5 \times \frac{1}{4} + 0.5 \times \frac{2}{4} + 0.33 \times \frac{1}{4} = 0.46$. ∎

4.1.3 Structure Degree

If a keyword query $q = (D, C, \{w_1, \ldots, w_n\})$, there exists 2^n sub-queries, including the empty one. Each row in the database matches one of these sub-queries. If a row does not contain any query keyword, it matches the empty sub-query. Different sub-queries may have different importance. Intuitively, longer sub-queries are more important than shorter ones. A group-by is good if its covered rows match many important sub-queries.

We use a structure degree to measure whether important sub-queries (structures) appear frequently in the minimal answers. If a group-by has a high structure degree, it means that important sub-queries (structures) appear frequently in this group-by, and thus this group-by should be ranked high in the search engine.

Definition 5 (Structure Degree)**.** *Given a query Q, the sub-queries of Q are $\{c_1, \ldots, c_y\}$, the* ***structure degree*** *of a group-by g is defined as*

$$StructureDegree(g) = StructureDegree(Cov(g)) = \sum_{j=1}^{y} \frac{|c_j|}{|Q|} \times \frac{M_j}{|Cov(g)|} \quad (3)$$

where M_j is the number of rows in $Cov(g)$ that contain the sub-query c_j. ■

Since we assume that longer sub-queries are more important than shorter ones, we can use $\frac{|c_j|}{|Q|}$ to measure the importance of a sub-query c_j. Also, we use $\frac{M_j}{|Cov(g)|}$ to measure the weight of c_j in the group-by g, thus the score of c_j in group-by g can be measured by using $\frac{|c_j|}{|Q|} \times \frac{M_j}{|Cov(g)|}$.

The structure degree of a group-by g is calculated using the information in its covered rows ($Cov(g)$). Thus, $StructureDegree(g)$ and $StructureDegree(Cov(g))$ are the same.

Example 8 (Structure Degree)**.** *Continued from Example 6, suppose the search engine returns two group-bys (g and g') for the query $(D, C, \{Austin, Boston, 2001\})$. For group-by $g = (*, *, 2001, accessories, *)$, its covered rows match the following sub-queries: $(D, C, \{Boston, 2001\})$, $(D, C, \{Austin, 2001\})$, and $(D, C, \{2001\})$. For group-by $g' = (*, *, *, *, 43)$, its covered rows match the following sub-queries: $(D, C, \{Boston, 2001\})$, $(D, C, \{Austin\})$, and $(D, C, \{Boston\})$.*

In Figure 5, the number of rows covered by group-by g is $|Cov(g)| = 3$, the number of rows in $Cov(g)$ that match $(D, C, \{Boston, 2001\})$ is $M_1 = 1$, the number of rows in $Cov(g)$ that match $(D, C, \{Austin, 2001\})$ is $M_2 = 1$ and the number of rows in $Cov(g)$ that match $(D, C, \{2001\})$ is $M_3 = 1$. So, the structure degree of group-by g is $StructureDegree(g) = \frac{2}{3} \times \frac{1}{3} + \frac{1}{3} \times \frac{1}{3} + \frac{2}{3} \times \frac{1}{3} = 0.56$. Similarly, the number of rows covered by group-by g' is $|Cov(g')| = 4$, the number of rows in $Cov(g')$ that match $(D, C, \{Boston, 2001\})$ is $M_1 = 1$, the number of rows in $Cov(g')$ that match $(D, C, \{Austin\})$ is $M_2 = 1$ and the number of rows in $Cov(g')$ that match $(D, C, \{Boston\})$ is $M_3 = 1$. So, the structure degree of group-by g' is $StructureDegree(g') = \frac{1}{3} \times \frac{1}{4} + \frac{1}{3} \times \frac{1}{4} + \frac{2}{3} \times \frac{1}{4} = 0.33$. ■

4.1.4 The Overall Scoring Function

Let g be the max-join result of group-bys g_1 and g_2. The scores of group-by g can be calculated using the information in $Cov(g_1) \cup Cov(g_2)$. The overall score of group-by g is the linear combination of its density score, dedication score and structure degree, that is,

$$\begin{aligned} &Score(g) = Score(Cov(g_1) \cup Cov(g_2)) \\ &= e_1 \times Density(g) + e_2 \times Dedication(g) + (1 - e_1 - e_2) \times StructureDegree(g) \end{aligned}$$

where e_1, e_2 are two coefficients, $0 \leq e_1, e_2 \leq 1$.

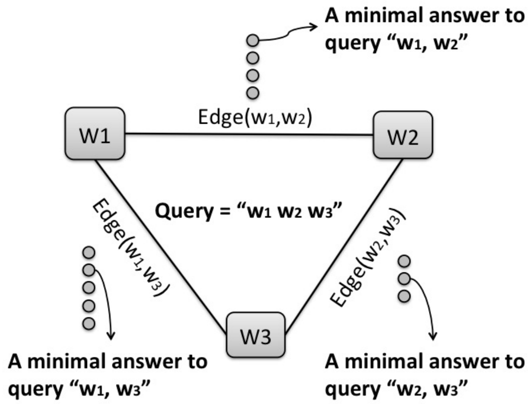


Figure 6: An example of Query Keyword Graph in Chapter 4

4.2 Top-k Query Processing

At the beginning of the query processing, a query keyword graph is constructed by using the IPJ index. For example, if the query q is $(D, C, \{w_1, w_2, w_3\})$, the corresponding query keyword graph is shown in Figure 6. Each vertex in the graph represents a query keyword and each edge contains a set of corresponding minimal answers.

Other steps of the query processing are the same with the keyword graph approach [22]. We need to check $|q| - 1 = 3 - 1 = 2$ edges (ignoring the edge with the largest number of minimal answers) in the graph to generate all the candidate answers. Then, we delete duplicate, empty or non-minimal group-bys in the candidate answers. In our example, we need to check edges (w_1, w_2) and (w_2, w_3). The edge (w_1, w_3) is ignored and does not need to be checked since it has more minimal answers than the other edges.

To answer query $q = (D, C, \{w_1, \ldots, w_m\})$, using Theorem 1 repeatedly, we only need to check $m - 1$ edges covering all keywords $w_1, \ldots, w_m$ in the clique. Each edge is associated with the set of minimal answers to a query on a pair of keywords. The weight of the edge is the size of the answer set. In order to reduce the total cost of the joins, heuristically, we can find a spanning tree connecting the m keywords such that the product of the weights on the edges is minimized.

If t is a minimal answer to aggregate keyword query $Q = (D, C, \{w_1, \cdots, w_m\})$, then there exists minimal answers t_1 and t_2 to queries $(D, C, \{w_1, w_2\})$ and $(D, C, \{w_e, \cdots, w_m\})$, respectively, such that $t = t_1 \vee t_2$.

In the query keyword graph, each edge is associated with a set of minimal answers. We use a **node** to represent a minimal answer of the corresponding edge, as shown in Figure 6. All nodes are **black nodes** at the beginning. As shown in Figure 7, each **line** represents a max-join operation on two black nodes. The max-join result is a candidate answer. We need to perform 12 max-join operations in order to generate all the candidate answers. Our top-k method detects some black

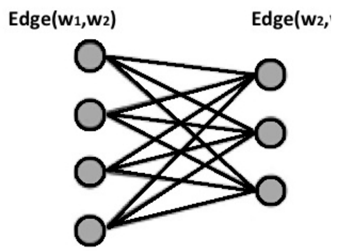


Figure 7: At the beginning all nodes are black

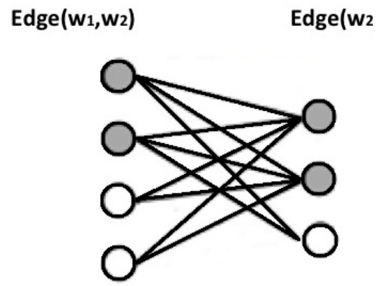


Figure 8: Detection in the bounding step

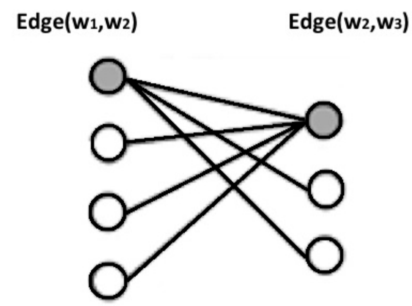


Figure 9: Further pruning in the pruning step

nodes as **white nodes** (Figure 8), such that if max-joins are all on white nodes, the corresponding max-join results are not top-k answers.

We have to do many max-join operations if generating all minimal answers. Since we only need top-k answers, some unnecessary max-join operations can be pruned.

We develop a two-step (the bounding step and the pruning step) pruning method to prune unnecessary max-join operations. In Figure 7, each node represents a minimal answer in the corresponding edge. All these nodes are black at the beginning. To prune unnecessary joins, the bounding step detects some black nodes as white nodes (Figure 8), such that if max-joins are all on white nodes, the corresponding max-join results are not top-k answers. The pruning step is developed to help detect more white nodes in the checked edges (Figure 9). The number of max-joins reduced by half after using these two steps. We only need to perform 6 max-join operations (max-joins that are all on white nodes are pruned). The more white nodes we detect, the more max-join operations we can prune.

4.2.1 The Bounding Step

Suppose there are n edges in the query keyword graph and thus we need to check $n-1$ edges to generate all the candidate answers. To generate one candidate answer g, we need to perform max-joins on a set of nodes $\{A_1, \ldots, A_{n-1}\}$, where A_i is a node from a corresponding checked edge. As mentioned in Section 4.1.4, the overall score of g is calculated by using information in $Cov(A_1) \cup Cov(A_2) \cup \cdots \cup Cov(A_{n-1})$, so we can define an upper bound for g using the overall scores of those nodes, as shown in Equation 4.5. If the upper bound is smaller than a threshold s, we do not need to perform max-join operations on these nodes. To find a suitable threshold, we generate k answers (may not be top-k answers) and calculate their overall scores. We use the lowest overall score as the threshold.

$$UpperBound(g) = UpperBound(A_1, \ldots, A_{n-1}) = \sum_{i}^{n-1} Score(A_i)$$

where $Score(A_i)$ is the overall score of node A_i.

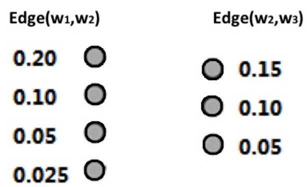


Figure 10: Sort the nodes for each edge

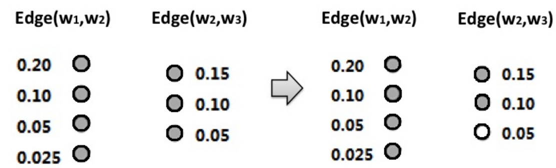


Figure 11: Detect white nodes for edge(w_2, w_3)

Example 9 (The bounding step). *Suppose the query is* $(D, C, \{w_1, w_2, w_3\})$ *and the corresponding query keyword graph is as shown in Figure 6. To generate candidate answers, we need to check edge* (w_1, w_2) *and edge* (w_2, w_3) *(Figure 7). All nodes of the checked edges are black at the beginning. To prune unnecessary max-join operations, we then detect some white nodes according to the following steps.*

First, we calculate the overall scores of all nodes and rank them according to their overall scores, as shown in Figure 10.

Second, we detect the white nodes for edge (w_2, w_3).

- *For each checked edge, we scan its associated nodes and find the black node with the lowest overall score. If the edge is not* (w_2, w_3), *we record the overall score of that black node in a set* S. *In our example,* $S = \{0.025\}$.
- *We scan every black node of edge* (w_2, w_3) *from top to down. Once we find a certain black node (suppose its overall score is* s'*), such that* $UpperBound(0.025, s')$ *is smaller than the threshold* s, *we stop scanning and mark that black node and nodes blow as white nodes. In our example,* $s' = 0.05$, *and the result is shown in Figure 11.*

Finally, we detect the white nodes for edge (w_1, w_2).

- *For each checked edge, we scan its associated nodes and find the black node with the lowest overall score. If the edge is not* (w_1, w_2), *we record the overall score of that black node in a set* S. *In our example,* $S = \{0.10\}$.
- *We scan every black node of edge* (w_1, w_2) *from top to down. Once we find a certain black node (suppose its overall score is* s'*), such that* $UpperBound(0.10, s')$ *is smaller than the threshold* s, *we stop scanning and mark that black node and nodes blow as white nodes. In our example,* $s' = 0.05$, *and the result is shown in Figure 12.* ■

The bounding step can detect many white nodes if we can find tight upper bounds. The limitation of this step is that the best upper bounds we can find are still not tight enough.

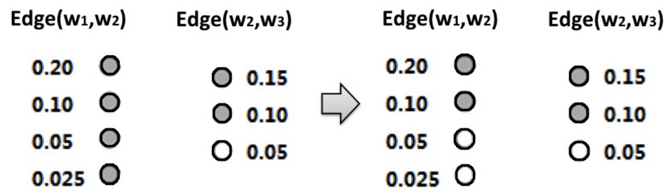


Figure 12: Detect white nodes for edge (w_1, w_2)

4.2.2 Tight Upper Bounds

Since the overall score is a linear combination of the three kinds of scores we defined (density, dedication, structure degree), we have the following equation:

$$\begin{aligned} UpperBound_{OverallScore}(g) &= e_1 \cdot UpperBound_{DensityScore}(g) + e_2 \cdot UpperBound_{DedicationScore}(g) \\ &\quad +(1 - e_1 - e_2) \cdot UpperBound_{StructureDegree}(g) \end{aligned}$$

We can obtain the following.

Theorem 2. *Let g be the max-join result of nodes (group-bys) A_1 and A_2. The upper bound of the density score of g is*

$$\frac{(Density(A_1) + Density(A_2) - 2 \times Density(A_1) \times Density(A_2))}{1 - Density(A_1) \times Density(A_2)},$$

the upper bound of the dedication score of g is $Dedication(A_1) + Dedication(A_2)$, and the upper bound of structure degree of g is $StructureDegree(A_1) + StructureDegree(A_2)$. The upper bounds are reachabel.

Proof. We only show the upper bound of the density score. The other two upper bounds can be proved similarly.

Suppose

- there are M rows in $Cov(A_1) \cup Cov(A_2)$, the density scores of these rows are $d_1, \ldots, d_M$;
- there are N' rows in $Cov(A_1) - Cov(A_1) \cup Cov(A_2)$, the density scores of these rows are $a_1, \ldots, a_{N'}$; and
- there are N'' rows in $Cov(A_2) - Cov(A_1) \cup Cov(A_2)$, the density scores of these rows are $b_1, \ldots, b_{N''}$.

For simplicity, we assume that each row has the same length (number of keywords). We have $Density(A_1) = \frac{\sum_{i=1}^{N'} a_i + \sum_{i=1}^{M} d_i}{N'+M}$, $Density(A_2) = \frac{\sum_{i=1}^{N''} b_i + \sum_{i=1}^{M} d_i}{N''+M}$, and $Density(g) = \frac{\sum_{i=1}^{N'} a_i + \sum_{(} i=1)^{N''} b_i + \sum_{i=1}^{M} d_i}{N'+N''+M}$. We can prove the upper bound once we show the following.

Lemma 1. *The upper bound of $Density(g)$ is:* $\frac{(Density(A_1)+Density(A_2)-2\times Density(A_1)\times Density(A_2))}{1-Density(A_1)\times Density(A_2)}$

Proof. Since each density score is in range $[0,1]$, we have: $0 \leq B = \sum_{i=1}^{N'} a_i \leq N'$, $0 \leq C = \sum_{i=1}^{N''} b_i \leq N''$, and $0 \leq D = \sum_{i=1}^{M} d_i \leq M$.

Let ξ be a very small positive number, and let $D' = D - \xi, B' = B + \xi, C' = C + \xi$, so we have

$$Density(A_1) = \frac{\sum_{i=1}^{N'} a_i + \sum_{i=1}^{M} d_i}{N' + M} = \frac{B + D}{N' + M} = \frac{B' + D'}{N' + M}$$
$$Density(A_2) = \frac{\sum_{i=1}^{N''} b_i + \sum_{i=1}^{M} d_i}{N'' + M} = \frac{C + D}{N'' + M} = \frac{C' + D'}{N'' + M}$$
$$\frac{D' + B' + C'}{N' + N'' + M} = \frac{D + B + C + \xi}{N' + N'' + M} > \frac{D + B + C}{N' + N'' + M} = Density(g)$$

If D becomes smaller (or B and C become larger), $Density(g)$ would become larger. So, if the upper bound of $Density(g)$ is reached, D must be 0, which means the rows covered by both A_1 and A_2 contain no query keywords.

Since D is 0, we have $Density(g) = \frac{B+C}{N'+N''+M}$, $Density(A_1) = \frac{B}{N'+M}$, and $Density(A_2) = \frac{C}{N''+M}$.

Let $M' = M + \xi, N_1' = N' - \xi, N_1'' = N'' - \xi$. We have:

$$\frac{B + C}{N_1' + N_1'' + M'} = \frac{B + C}{N' + N'' + M - \xi} > \frac{B + C}{N' + N'' + M} = Density(g)$$
$$0 \leq B = \sum_{i=1}^{N'} a_i \leq N'$$
$$0 \leq C = \sum_{i=1}^{N''} b_i \leq N''$$

If N' and N'' become smaller(or M becomes larger), $Density(g)$ would become larger. So, if the upper bound of $Density(g)$ is reached, N' must be B and N'' must be C, which means $a_1 = \cdots = a_{N'} = 1$ and $b_1 = \cdots = b_{N''} = 1$.

So, the upper bound of $Density(g)$ is reached if $D = 0, B = N'$ and $C = N''$. In such a case, the upper bound of $Density(g)$ is

$$\frac{(Density(A_1) + Density(A_2) - 2 \times Density(A_1) \times Density(A_2))}{1 - Density(A_1) \times Density(A_2)}$$

The above upper bounds are reached in the case shown in Figure 13. ■

4.2.3 The Pruning Step

As discussed earlier, the bounding step may not be able to find all white nodes, so we use the pruning step to detect more white nodes.

In the pruning step, we define a score function $f(C) = (Score(C) - s) \times |C|$, where C represents a set of rows, $Score()$ is the overall score function we defined above, and s is the threshold.

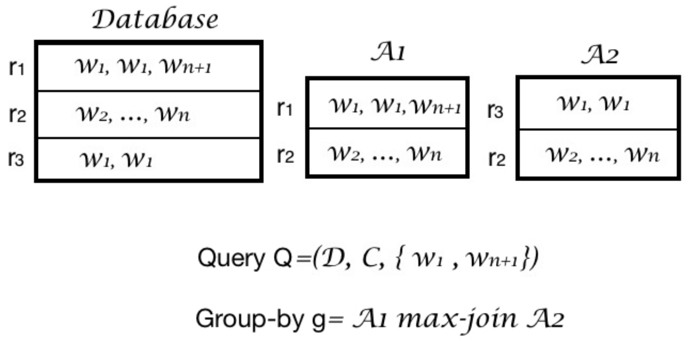


Figure 13: An example when the upper bounds are reached

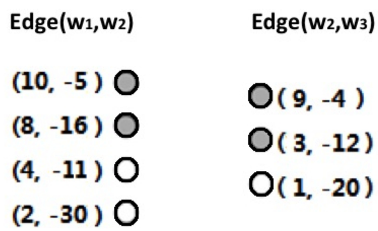


Figure 14: Define two types of scores for each node

The candidate answer g is generated by performing max-joins on a set of nodes $\{A_1, \ldots, A_{n-1}\}$, where A_i is a minimal answer of a corresponding edge. So, for each node A_i, its covered rows $Cov(A_i)$ can be divided into two parts, $Cov(A_i)_1$ and $Cov(A_i)_2$, such that, for each row r in $Cov(A_i)_1$, $Score(r) \leq s$; and, for each row r' in $Cov(A_i)_2$, $Score(r') < s$. Therefore, we can calculate another two types of scores $\Big(f(Cov(A_i)_1)$ and $f(Cov(A_i)_2)\Big)$ for each node A_i using the function f. Figure 14 is an example about the two types of scores of each node of the checked edges.

Theorem 3. *Let s be the threshold on the overall score of k generated answers. Given a group-by g, if $f(Cov(g)_1) + f(Cov(g)_2) < 0$, the overall score of g is smaller than the threshold s.*

Theorem 4. *Let s be the threshold on the overall score of k generated answers. Suppose the candidate answer g is generated by performing max-joins on a set of nodes (minimal answers) $\{A_1, \ldots, A_{n-1}\}$, where A_i is a minimal answer of a corresponding checked edge in the query keyword graph. If the following inequality is satisfied, the overall score of g is smaller than the threshold s.*

$$\sum_{i=1}^{n-1} f(Cov(A_i)_1) + \min\{f(Cov(A_1)_2), \ldots, f(Cov(A_{n-1})_2)\} < 0$$

Proof. We need to prove that given a group-by g, if $f(Cov(g)_1) + f(Cov(g)_2) < 0$, the overall score of g is smaller than the threshold s.

According to the definition of function f, we have:

$$f(Cov(g)_1) = (Score(Cov(g)_1) - s) \times |Cov(g)_1|$$
$$f(Cov(g)_2) = (Score(Cov(g)_2) - s) \times |Cov(g)_2|$$

We already know that $f(Cov(g)_1) + f(Cov(g)_2) < 0$. So, we have

$$\begin{aligned}
&0 > f(Cov(g)_1) + f(Cov(g)_2) \\
&= (Score(Cov(g)_1) - s) \times |Cov(g)_1)| + (Score(Cov(g)_2) - s) \times |Cov(g)_2| \\
&= Score(Cov(g)_1) \times |Cov(g)_1| + Score(Cov(g)_2) \times |Cov(g)_2| - s \times (|Cov(g)_1| + |Cov(g)_2|) \\
&= Score(Cov(g)_1) \times |Cov(g)_1| + Score(Cov(g)_2) \times |Cov(g)_2| - s \times |Cov(g)| \\
&s \times |Cov(g)| > Score(Cov(g)_1) \times |Cov(g)_1| + Score(Cov(g)_2) \times |Cov(g)_2| \\
&s > \frac{Score(Cov(g)_1) \times |Cov(g)_1| + Score(Cov(g)_2) \times |Cov(g)_2|}{|Cov(g)|}
\end{aligned}$$

So we only need to prove the following equation:

$$Score(Cov(g)) = \frac{Score(Cov(g)_1) \times |Cov(g)_1| + Score(Cov(g)_2) \times |Cov(g)_2|}{|Cov(g)|}$$

For simplicity, we assume that each row has the same length l (number of keywords), and we have:

$$\begin{aligned}
&Density(Cov(g)) \\
&= \frac{Density(Cov(g)_1) \times Num(Cov(g)_1) + Density(Cov(g)_2) \times Num(Cov(g)_2)}{Num(Cov(g))} \\
&= \frac{Density(Cov(g)_1) \times |Cov(g)_1| \times l + Density(Cov(g)_2) \times |Cov(g)_2| \times l}{|Cov(g)| \times l} \\
&= \frac{Density(Cov(g)_1) \times |Cov(g)_1| + Density(Cov(g)_2) \times |Cov(g)_2|}{|Cov(g)|}
\end{aligned}$$

$$Dedication(Cov(g)) = \sum_{i=1}^{n} IDF(w_i) \times \frac{N_i}{|Cov(g)|} = \frac{\sum_{i=1}^{n} IDF(w_i) \times N_i}{|Cov(g)|}$$

$$= \frac{\sum_{i=1}^{n} IDF(w_i) \times (N_i' + N_i'')}{|Cov(g)|} = \frac{\sum_{i=1}^{n} IDF(w_i) \times N_i' + sum_{i=1}^{n} IDF(w_i) \times N_i''}{|Cov(g)|}$$

$$= \frac{|Cov(g)_1| \times \sum_{i=1}^{n} IDF(w_i) \times \frac{N_i'}{|Cov(g)_1|} + |Cov(g)_2| \times sum_{i=1}^{n} IDF(w_i) \times \frac{N_i''}{|Cov(g)_2|}}{|Cov(g)|}$$

$$= \frac{|Cov(g)_1| \times Dedication(Cov(g)_1) + |Cov(g)_2| \times Dedication(Cov(g)_2)}{|Cov(g)|}$$

$$StructureDegree(Cov(g)) = \sum_{j=1}^{y} \frac{|c_j|}{|Q|} \times \frac{M_j}{|Cov(g)|} = \frac{\sum_{j=1}^{y} \frac{|c_j|}{|Q|} \times M_j}{|Cov(g)|} = \frac{\sum_{j=1}^{y} \frac{|c_j|}{|Q|} \times (M_j' + M_j'')}{|Cov(g)|}$$

$$= \frac{\sum_{j=1}^{y} \frac{|c_j|}{|Q|} \times M_j' + \sum_{j=1}^{y} \frac{|c_j|}{|Q|} \times M_j''}{|Cov(g)|}$$

$$= \frac{|Cov(g)_1| \times \sum_{j=1}^{y} \frac{|c_j|}{|Q|} \times \frac{M_j'}{|Cov(g)_1|} + |Cov(g)_2| \times \sum_{j=1}^{y} \frac{|c_j|}{|Q|} \times \frac{M_j''}{|Cov(g)_2|}}{|Cov(g)|}$$

$$= \frac{|Cov(g)_1| \times StructureDegree(Cov(g)_1) + |Cov(g)_2| \times StructureDegree(Cov(g)_2)}{|Cov(g)|}$$

where w_i is a query keyword, N_i represents the number of rows that contain w_i in Cov(g), N_i' is the number of rows that contain w_i in $Cov(g)_1$, N_i'' is the number of rows that contain w_i in $Cov(g)_2$, c_j is a sub-query, M_j represents the number of rows that contain c_j in Cov(g), M_i' is the number of rows that contain c_j in $Cov(g)_1$, and M_i'' is the number of rows that contain c_j in $Cov(g)_2$.

Since the overall score is the linear combination of density score, dedication score and structure degree, we have:

$$
\begin{aligned}
&Score(Cov(g)) = e_1 \times Density(Cov(g)) + e_2 \times Dedication(Cov(g))\\
&+(1-e_1-e_2) \times StructureDegree(Cov(g))\\
&= e_1 \times (\frac{Density(Cov(g)_1) \times |Cov(g)_1| + Density(Cov(g)_2) \times |Cov(g)_2|}{|Cov(g)|})\\
&+e_2 \times (\frac{|Cov(g)_1| \times Dedication(Cov(g)_1) + |Cov(g)_2| \times Dedication(Cov(g)_2)}{|Cov(g)|})\\
&+(1-e_1-e_2) \times \frac{1}{|Cov(g)|}\\
&\times(|Cov(g)_1| \times StructureDegree(Cov(g)_1) + |Cov(g)_2| \times StructureDegree(Cov(g)_2))\\
&= \frac{1}{|Cov(g)|} \times \Bigg[\Big(e_1 \times Density(Cov(g)_1) + e_2 \times Dedication(Cov(g)_1)\\
&+(1-e_1-e_2) \times StructureDegree(Cov(g)_1)\Big) \times |Cov(g)_1|\\
&+\Big(e_1 \times Density(Cov(g)_2) + e_2 \times Dedication(Cov(g)_2)\\
&+(1-e_1-e_2) \times StructureDegree(Cov(g)_2)\Big) \times |Cov(g)_2|\Bigg]\\
&= \frac{Score(Cov(g)_1) \times |Cov(g)_1| + Score(Cov(g)_2) \times |Cov(g)_2|}{|Cov(g)|} < s
\end{aligned}
$$

We need to prove that : suppose the candidate answer g is generated by performing max-joins on a set of nodes (minimal answers) $\{A_1, \ldots, A_{n-1}\}$, each of which is from a corresponding checked edge in the query keyword graph. If $\sum_{i=1}^{n-1} f(Cov(A_i)_1) + \min\{f(Cov(A_1)_2), \ldots, f(Cov(A_{n-1})_2)\} < 0$, the overall score of g is smaller than the threshold s.

We first prove the case for $n = 3$. The candidate answer g is generated by performing max-joins on a set of nodes (minimal answers) $\{A_1, A_2\}$, each of these nodes is from a corresponding checked edge in the query keyword graph. We need to prove that: if $f(Cov(A_1)_1) + f(Cov(A_2)_1) + \min\{f(Cov(A_1)_2), f(Cov(A_2)_2)\} < 0$, the overall score of g is smaller than the threshold s.

Since g is generated by performing max-joins on A_1 and A_2, as we discussed in Section 4.1.4, the scores of group-by g will be calculated using information in $Cov(A_1) \cup Cov(A_2)$. So we have:

$$
\begin{aligned}
f(Cov(g)_1) &= f(Cov(A_1)_1 \cup Cov(A_2)_1) \leq f(Cov(A_1)_1) + f(Cov(A_2)_1)\\
f(Cov(g)_2) &= f(Cov(A_1)_2 \cup Cov(A_2)_2) \leq \min\{Cov(A_1)_2, Cov(A_2)_2\}
\end{aligned}
$$

So we have:

$$
\begin{aligned}
&f(Cov(g)_1) + f(Cov(g)_2)\\
&\leq f(Cov(A_1)_1) + f(Cov(A_2)_1) + min\{Cov(A_1)_2, Cov(A_2)_2\} < 0
\end{aligned}
$$

According to Theorem 3, the overall score of g is smaller than the threshold s. Similarly, we can prove the case for $n \geq 4$:

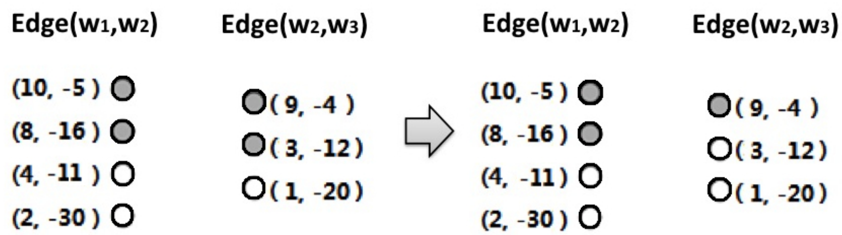


Figure 15: Detect white nodes for edge(w_2, w_3)

$$f(Cov(g)_1) = f(Cov(A_1)_1 \cup \cdots \cup Cov(A_2)_1) \leq f(Cov(A_1)_1) + \cdots + f(Cov(A_2)_1)$$
$$f(Cov(g)_2) = f(Cov(A_1)_2 \cup \cdots \cup Cov(A_2)_2) \leq \min\{Cov(A_1)_2, \ldots, Cov(A_2)_2\}$$
$$f(Cov(g)_1) + f(Cov(g)_2) \leq f(Cov(A_1)_1) + \cdots + f(Cov(A_2)_1) + min\{Cov(A_1)_2, \ldots, Cov(A_2)_2\} < 0$$

■

Example 10 (The pruning step)**.** *In the scenario of the above example, two white nodes of edge* (w_1, w_2) *and one white node of edge* (w_2, w_3) *are detected in the bounding step (Figure 12). In the pruning step, more white nodes can be detected using Theorem 4.*

First, we detect more white nodes for edge (w_2, w_3)*.*

- *Create two sets,* S_1 *and* S_2*.*
- *For each checked edge, if the edge is not* (w_2, w_3) *and suppose its associated white nodes are* $B_1, \ldots, B_h$*, 1) we scan these white nodes and record* $\max\{f(Cov(B_1)_1), f(Cov(B_2)_1), \cdots, f(Cov(B_h)_1)\}$ *in* S_1*; and 2) we also record* $\max\{f(Cov(B_1)_2), f(Cov(B_2)_2), \cdots, f(Cov(B_h)_2)\}$ *in* S_2*. In our example,* $S_1 = \{4\}$*,* $S_2 = \{-11\}$*.*
- *Let* s_1 *be the sum of items in* S_1 *and* s_2 *be the minimal item in* S_2*. In our example,* $s_1 = 4$ *and* $s_2 = -11$*.*
- *Scanning every black node of edge* (w_2, w_3) *from top to down. Once we find a certain black node* z*, such that* $s_1 + f(Cov(z)_1) + \min\{s_2, f(Cov(z)_2)\} < 0$ *(**Corollary 1**), we stop scanning and mark that black node and nodes blow as white nodes. In our example,* $f(Cov(z)_1) = 3$ *and* $f(Cov(z)_2) = -12$*, the result is shown in Figure 15.*

Second, we detect more white nodes for edge (w_1, w_2)*.*

- *Create two sets,* S'_1 *and* S'_2*.*

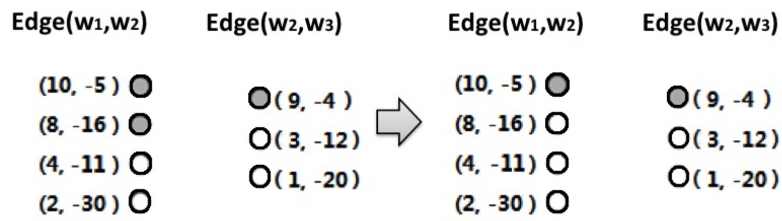


Figure 16: Detect white nodes for edge(w_2, w_3)

- *For each checked edge, if the edge is not* (w_1, w_2) *and suppose its associated white nodes are* $B'_1, \dots, B'_{h'}$*, 1)we scan these white nodes and record* $\max\{f(Cov(B'_1)_1), f(Cov(B'_2)_1), \cdots, f(Cov(B'_{h'})_1)\}$ *in* S'_1*; (2) we also record* $\max\{f(Cov(B'_1)_2), f(Cov(B'_2)_2), \dots, f(Cov(B'_{h'})_2)\}$ *in* S'_2*. In our example,* $S'_1 = \{3\}$*,* $S'_2 = \{-12\}$*.*
- *Let* s'_1 *be the sum of items in* S'_1 *and* s'_2 *be the minimal item in* S'_2*. In our example,* $s'_1 = 3$ *and* $s'_2 = -12$*.*
- *Scanning every black node of edge* (w_1, w_2) *from top to down. Once we find a certain black node* z'*, such that* $s'_1 + f(Cov(z')_1) + \min\{s'_2, f(Cov(z')_2)\} < 0$ *(Theorem 4), we stop scanning and mark that black node and nodes blow as white nodes. In our example,* $f(Cov(z)_1) = 8$ *and* $f(Cov(z)_2) = -16$*, the result is shown in Figure 16.* ■

In the pruning step, we detect more white nodes for both edge (w_1, w_2) and edge (w_2, w_3). The number of max-joins reduced by half after using the bounding step and the pruning step.

5 Experimental Results

In this section, we report an empirical study of our top-k query answering method on two real data sets. We first describe the user study which is used to learn the coefficients for the overall scoring function. Then, we report the effectiveness of the bounding step and the pruning step. Finally, we evaluate the top-k query answering method and the complete query answering method under various number of tuples and dimensions.

5.1 Setup and Data Sets

All the experiments were conducted on a PC computer running the Microsoft Windows 7 Professional Edition operating system, with a 2.4 GHz CPU, 2.0 GB main memory, and a 250 GB hard disk. The programs were implemented in JAVA and were compiled using eclipse.

The e-Fashion dataset and the SuperstoreSales dataset have been used in the projects of SAP Research on keyword search on relational databases. Since our project is supported by SAP Research, we use these two datasets to empirically evaluate our aggregate keyword search methods.

Attribute	Description
Store name	branch store name
State	which State the branch store is located
City	which city the branch store is located
Year	year of the sales information
Quarter	quarter of the sales information
Month	month of the sales information
Lines	type of the product sold in the branch store
Sales revenue	sales revenue of the product
Quantity sold	quantity sold of the product

Table 9: Dimensions of the e-Fashion database

The dimensions of the e-Fashion dataset are shown in Table 9. There are 9 dimensions, 4300 tuples and 4000 unique keywords in the e-Fashion dataset. The SuperstoreSales dataset has 21 dimensions, 8339 tuples and 0.35 million unique keywords. Table 10 shows the dimensions in the SuperstoreSales dataset. To keep our discussion simple, we assume all the database fields are text attributes. In data representation, we adopted the popular packing technique [3]. A value on a dimension is mapped to an integer. The star value on a dimension is mapped to 0. We also map keywords to integers.

5.2 User Study

We use the traditional linear regression model [10, 8] to learn the ranking function. A user study is then performed to calculate the coefficients of the overall scoring function. For each tested query, we randomly select 5 answers for users to evaluate. For each selected answer x_i, its density score (x_{i1}), dedication score (x_{i2}) and structure degree (x_{i3}) are pre-calculated. Let y_i be the score evaluated by users for the answer x_i, we have the following linear regression model.

$$f(x_i) = e_1 \times x_{i1} + e_2 \times x_{i2} + (1 - e_1 - e_2) \times x_{i3}$$

The minimum sum of squares (SSE, the error sum of squares) we used in the learning model is $SSE = \sum_{i=1}^{m}(y_i - f(x_i))^2$, where m is the total number of selected answers evaluated by users.

In the user study, we designed three types of tested queries, each of which represents a possible search intension. For example, given a query $Q = (D, C, \{w_1, w_2, w_3\})$, it may have the following search intensions:

1. "w_1 **or** w_2 **or** w_3" (Table 11)
2. "w_1 **and** w_2 **and** w_3" (Table 13)
3. Others, i.e. "w_1 **and** w_2 **OR** w_1 **and** w_3" (Table 15)

Attribute	Description
Order ID	ID of the order
Order Date	the order date
Order Priority	priority of the order
Order Quantity	product quantity of the order
Sales	total price of the order
Discount	discount on the order
Ship Mode	ship method of the order
Profit	profit of the order
Unit Price	price per unit
Shipping Cost	cost of the shipping
Customer Name	name of the customer
Customer State	which State the customer is located
Zip Code	Zip code of the customer location
Region	region of the customer location
Customer Segment	customer type
Product Category	category of the product
Product Sub-Category	sub-category of the product
Product Name	name of the product
Product Container	container of the product
Product Base Margin	base margin of the product
Ship Date	shipping date

Table 10: Dimensions of the SuperstoreSales database

For each type of query, we test 10 instance queries. We have 10 people participating in the studies. We get 10 sets of results, each of which is from a single user and can be used to calculate a set of values of the coefficients. We also mix all the results from the users and get another set of values of the coefficients. So, we have 11 sets of values of the coefficients, as shown in Table 17 and Table 18.

The learning results may not be the best, since there are only 10 people in the user study and we only select 5 answers randomly for each tested query. We will get better coefficients if we have larger samples and more people.

5.3 Effectiveness of the Bounding Step and the Pruning Step

In the query keyword graph, each checked edge contains a set of minimal answers (black nodes). The bounding step and the pruning step prune many unnecessary max-joins by detecting some black nodes as white nodes for each checked edge. So, the effectiveness of the bounding step and the pruning step can be evaluated by measuring the rate of white nodes of the checked edges.

We test the following 6 queries, three of which are on the e-Fashion dataset and others are on

Query Template	Tested Queries
"w_1 **or** w_2 **or** w_3"	Q =(D, C, {Austin, Boston, Washington}) **Description** Each keyword represents a city. Users are interested in common information about these cities. For example, products sold in these cities. **Table 5.4** shows such an interesting result. **Keywords in other tested queries** "austin boston washington miami","sweaters trousers jackets", "paper envelopes tables bookcases", "michigan florida virginia maryland","newbury springs leighton", "2001 2002 2003", "sweaters trousers jackets outerwear","michigan florida virginia", "paper envelopes tables"

Table 11: Tested Queries 1

StoreName	City	Year	Quarter	Lines	QuantitySold
*	*	2003	*	accessories	78
e-Fashion Austin	Austin	2003	q1	accessories	78
e-Fashion Newbury	Boston	2003	q3	accessories	78
e-Fashion Tolbooth	Washington	2003	q3	accessories	78

Table 12: One good result for the query (D, C, {Austin, Boston, Washington})

the SuperstoreSales dataset. For each tested query, we measure the percentage of white nodes of the checked edge.

For the e-Fashion dataset,

$Q_1 = (D_{e-Fashion}, C_{e_{Fashion}}, \{Jackets, Leather, Sweaters, 2001\})$

$Q_2 = (D_{e-Fashion}, C_{e-Fashion}, \{Jackets, Leather, Sweaters\})$

$Q_3 = (D_{e-Fashion}, C_{e-Fashion}, \{2001, 2002, 2003, Jackets\})$

For the SuperstoreSales dataset,

$Q_4 = (D_{SuperstoreSales}, C_{SuperstoreSales}, \{Paper, Envelopes, Tables\})$

$Q_5 = (D_{SuperstoreSales}, C_{SuperstoreSales}, \{Roy, Matt, Collins\})$

$Q_6 = (D_{SuperstoreSales}, C_{SuperstoreSales}, \{Tracy, Truck, Box\})$

Figure 17 shows the experiment results on the e-Fashion dataset and Figure 18 is the results on the SuperstoreSales dataset. The bounding step is effective in detecting white nodes for Q_5. However, it detects few white nodes for Q_3. For Q_5, the bounding step can detect many white nodes because: 1) the overall scores of most group-bys are close to their upper bounds; and 2) the overall scores of most group-bys are much smaller than the threshold s. For Q_3, few white nodes

Query Template	**Tested Queries**
"w_1 **and** w_2 **and** w_3"	Q =(D, C, {php, html, ajax}) **Description** Each keyword represents a job skill, a job hunter is interested in jobs that contain as many related job skills as possible. **Table 5.6** shows such an interesting result. **Keywords in other tested queries** "tracy truck box","2001 austin trousers","2001 q1 trousers","express high furniture", "austin q1 trousers", "carolina express furniture", "austin q1 2001", "carolina high express","mobile android downtown"

Table 13: Tested Queries 2

JobDescription	Avg(USD)	JobType	Started	Location
*	*	*	*	richmond
... to add the necessary code into the current system to enable hotmail address to be used. I would love the user to also be ...	85	joomla, php, .net, ajax, software architecture	Nov.	richmond
... it needs to be fun and yet professional looking...	121	.net, ajax, html, graph design, website design	Oct.	richmond

Table 14: One good result for the query (D, C, {php, html, ajax})

are detected in the bounding step because: 1) the overall score of most group-bys are much smaller their upper bounds; or 2) the overall scores of most group-bys are larger than the threshold s. The pruning step is designed to detect more white nodes for each checked edge. After using the pruning step, we get better results. The pruning step detects many white nodes for all tested queries. For Q_3, about 90% of the nodes are detected as white nodes after the pruning step. For Q_2, although there is no big improvement after the pruning step, the result is still better than previous. In the pruning step, each group-by's covered tuples are divides into two types: 1) tuples with overall scores smaller than the threshold s and 2) tuples with overall scores not smaller than s. Such information can help better predicting if the overall score of a group-by is smaller than the threshold s.

Query Template	**Tested Queries**
"w_1 **and** w_3" **OR** "w_2 **and** w_3"	Q =(D, C, {roy, matt, collins} **Description** The first two keywords represent first names, the last keyword represents a last name. Users are interested in information about "roy collins" or "matt collins". **Table 5.8** shows such an interesting result. **Keywords in other tested queries** "sweaters trousers outerwear 2001","sweaters trousers newbury", "sweaters trousers outerwear newbury", "maryland georgia florida cleaner", "2001 2002 2003 newbury", "sweaters trousers 2001", "office supplies express air", "maryland georgia cleaner", "2001 2002 newbury"

Table 15: Tested Queries 3

OrderID	Priority	ShipMode	CustomerName	State	Container	Product
*	high	*	*	*	small box	laptop
130	high	regular air	roy collins	florida	small box	laptop
5318	high	expiress air	matt collins	michigan	small box	laptop

Table 16: One good result for the query (D, C, {roy, matt, collins})

5.4 The Top-k Query Answering Method and the Complete Query Answering Method

We use the e-Fashion dataset and the SuperstoreSales dataset to study the efficiency of the top-k query answering method. To study the scalability of our algorithm, we measure the query answering time of our method under various number of tuples and dimensions in the datasets.

We conduct two query answering experiments on the datasets. In our experiments, the top-k query answering method returns top-10 answers. In the first experiment, we change the number of tuples in the datasets. The corresponding results are shown in Figure 19 and Figure 20. For the complete query answering method, increasing the number of tuples results in a fairly linear increase in the runtime. One reason is that the number of max-join operations increases with the number of tuples. Another reason is that there could be more answers if the datasets contains more tuples. The top-k query answering method is also sensitive to the number of tuples in the datasets, but it is faster than the complete query answering method. The reason is that many unnecessary

Coefficients	No.1	No.2	No.3	No.4	No.5	No.6
e_1	16.869	16.207	16.418	18.014	18.135	15.757
e_2	24.440	20.884	24.920	23.910	32.815	24.111
e_3	4.500	5.095	4.788	4.868	4.669	5.276

Table 17: The user study results 1

Coefficients	No.7	No.8	No.9	No.10	Mix
e_1	14.925	15.037	17.137	19.475	16.765
e_2	26.524	27.383	30.775	34.009	26.453
e_3	5.226	5.352	4.783	3.970	4.867

Table 18: The user study results 2

max-join operations in the top-k query answering method are pruned after the bounding step and the pruning step. As the number of tuples increases, more unnecessary joins are pruned and the top-k query answering method performs better than the complete query answering method.

In the second experiment, we change the number of dimensions in the datasets. The corresponding results are shown in Figure 21 and Figure 22. The result of the second experiment is similar with that of the first experiment. When the number of dimensions increases, both the top-k query answering method and the complete query answering method spend longer time to find the answers. One reason is that when there are more dimensions in the datasets, the number of max-join operations does not increase but it takes longer time to perform each max-join operation. Another reason is that, as the dimensionality increases, more answers could be found. Thus more query processing time is needed for both methods, especially for the complete query answering method since it needs to find all the answers. In summary, our experimental results on the two datasets clearly show that the top-k query answering method is highly feasible.

5.5 The Effect of k

Figure 23 shows the runtime of the top-k query answering method on the two data sets with respect to k. Clearly, the smaller the value of k, the more efficient the results. As discussed in Chapter 5, at the beginning of top-k query answering process, we generate k answers (may not be top-k) and use the lowest overall score as the threshold. The larger the threshold is, the more max-join operations we can prune. If k becomes smaller, the threshold could become larger and thus we could prune more max-join operations.

From Figure 23, we find that results on the SuperstoreSales dataset are not sensitive to the value of k. The reverse is true for the e-Fashion dataset. One possibility is that the overall scores of answers on the SuperstoreSales dataset are very close, so even if k has a great increase in its value, the threshold does not have a great change and thus the runtime does not have a great increase. For the e-Fashion dataset, the top-k query answering method is more efficient than the complete

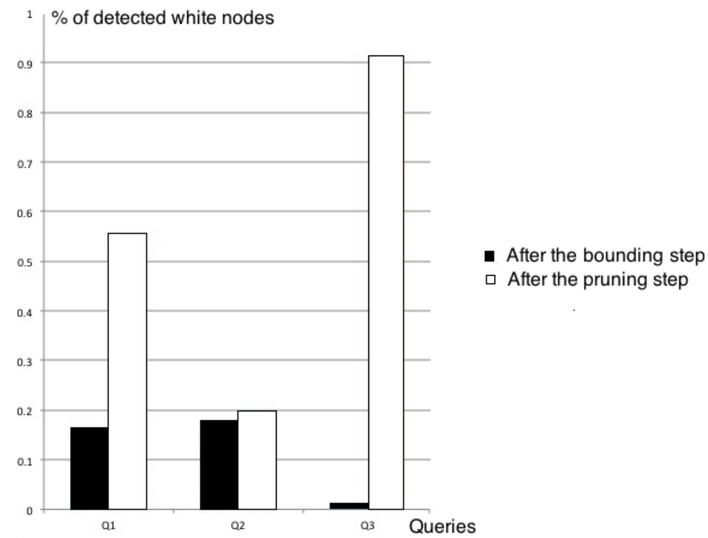


Figure 17: Effectiveness of the bounding step and the pruning step on the e-Fashion dataset

query answering method if the value of k is small (< 80). For the SuperstoreSales dataset, the top-k query answering method is more efficient than the complete query answering method for most values of k.

6 Conclusions

In this paper, we tackled two practical and interesting problems to improve the efficiency and effectiveness of aggregate keyword search on large relational databases. First, aggregate keyword search can be very costly on large relational databases, partly due to the lack of efficient indexes. To tackle this problem, we designed a new index which is efficient both in size and in constructing time. Second, finding the top-k answers to an aggregate keyword query has not been addressed systematically, including both the ranking model and the efficient evaluation methods. To tackle this problem, we proposed a general ranking model and an efficient ranking algorithm which using a two-step method to prune unnecessary max-join operations. We also reported a systematic performance evaluation using real data sets. Our experimental results show that our new index is very efficient and our two-step method is very effective. Our top-k query answering method can find top-k answers in a shorter time than that of the complete query answering method on the real data sets.

Our work on aggregate keyword search is focused on a single table. As future work, we plan to extend our work in multiple tables. Moreover, in some cases, a user may find a minimal answer that is close to the search intension, it could be interesting if we can help the user find other group-bys that are close to this minimal answer.

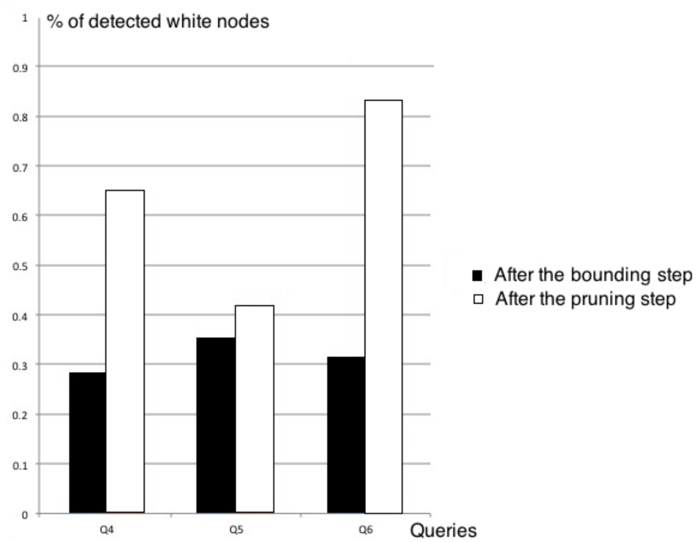


Figure 18: Effectiveness of the bounding step and the pruning step on the SuperstoreSales dataset

References

[1] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings of the 18th International Conference on Data Engineering, ICDE 2002, 26 February - 1 March 2002, San Jose, CA*, pages 5–16. IEEE Computer Society, 2002.

[2] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004*, pages 564–575. Morgan Kaufmann, 2004.

[3] K. S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proceedings ACM SIGMOD International Conference on Management of Data, SIGMOD 1999, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 359–370. ACM Press, 1999.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proceedings of the 18th International Conference on Data Engineering, ICDE 2002, 26 February - 1 March 2002, San Jose, CA*, pages 431–440. IEEE Computer Society, 2002.

[5] Y. Chen, W. Wang, and Z. Liu. Keyword-based search and exploration on databases. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 1380–1383. IEEE Computer Society, 2011.

[6] B. Ding, Y. Yu, B. Zhao, C. X. Lin, J. Han, and C. Zhai. Keyword search in text cube: Finding top-k aggregated cell documents. In *Proceedings of the 2010 Conference on Intelligent*

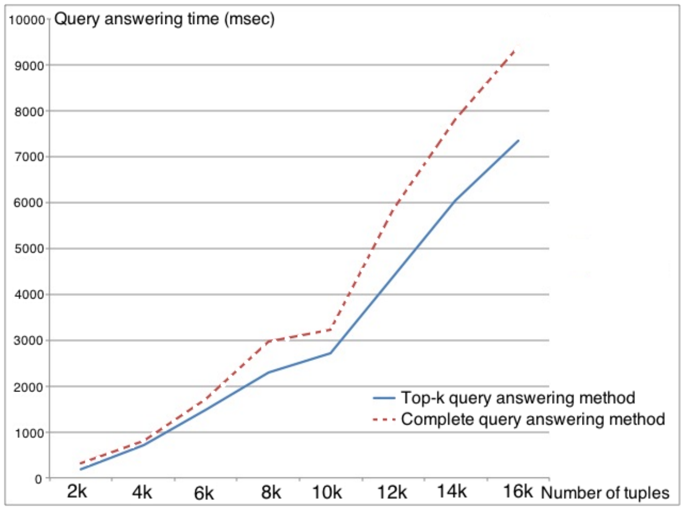


Figure 19: Efficiency of theTop-k query answering method and the complete query answering method on the e-Fashion dataset under various number of tuples

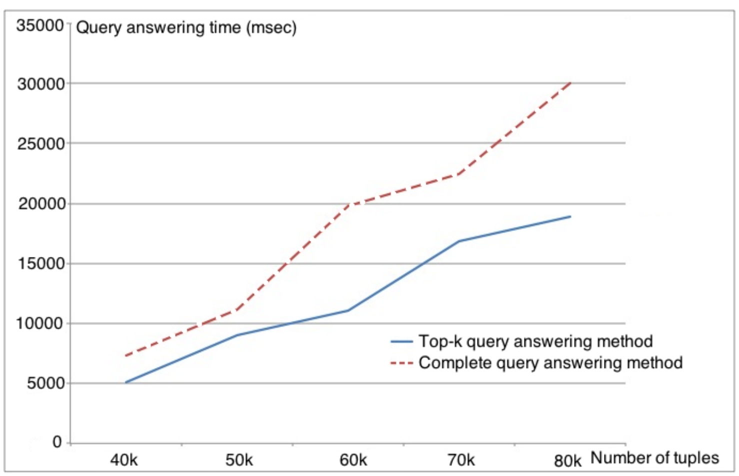


Figure 20: Efficiency of theTop-k query answering method and the complete query answering method on the SuperstoreSales dataset under various number of tuples

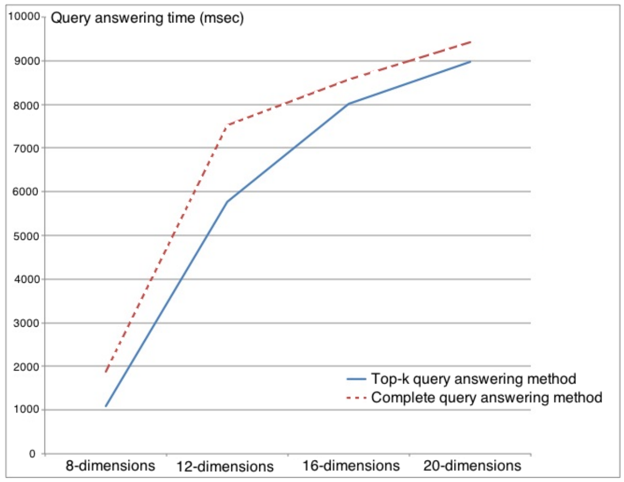


Figure 21: Efficiency of theTop-k query answering method and the complete query answering method on the e-Fashion dataset under various number of dimensions

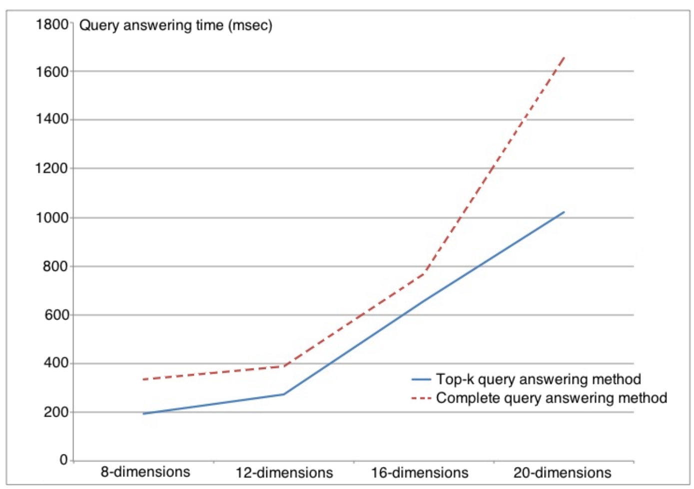


Figure 22: Efficiency of theTop-k query answering method and the complete query answering method on the SuperstoreSales dataset under various number of dimensions

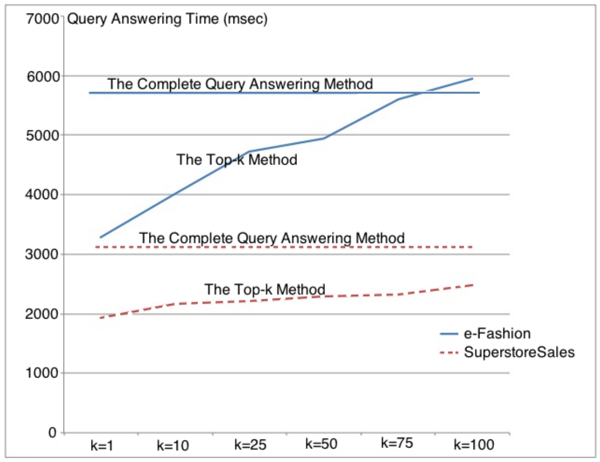


Figure 23: Effect of the parameter k on the e-Fashion and the SuperstoreSales datasets

Data Understanding, CIDU 2010, October 5-6, 2010, Mountain View, California, USA, pages 145–159. NASA Ames Research Center, 2010.

[7] B. Ding, B. Zhao, C. X. Lin, J. Han, and C. Zhai. Topcells: Keyword-based search of top-k aggregated documents in text cube. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 381–384. IEEE, 2010.

[8] N. R. Draper and H. Smith. *Applied regression analysis (2. ed.)*. Wiley series in probability and mathematical statistics. Wiley, 1981.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[10] J. Fedorowicz. Database evaluation using multiple regression techniques. In *Proceedings of Annual Meeting, SIGMOD 1984, Boston, Massachusetts, June 18-21, 1984*, pages 70–76. ACM Press, 1984.

[11] S. L. Hakimi. Steiner's problem in graphs and its implications. *Wiley Periodicals, Inc.*, 1(2):113–133, 1971.

[12] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, August 20-23, 2002, Hong Kong, China*, pages 670–681. Morgan Kaufmann, 2002.

[13] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

[14] J. Koren, Y. Zhang, and X. Liu. Personalized interactive faceted search. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 477–486. ACM, 2008.

[15] Z. Li, H. Xu, Y. Lu, and A. Qian. Aggregate nearest keyword search in spatial databases. In *Advances in Web Technologies and Applications, Proceedings of the 12th Asia-Pacific Web Conference, APWeb 2010, Busan, Korea, 6-8 April 2010*, pages 15–21. IEEE Computer Society, 2010.

[16] S. E. Robertson, S. Walker, and M. Hancock-Beaulieu. Okapi at trec-7: Automatic ad hoc, filtering, vlc and interactive. In *Text REtrieval Conference (TREC)*, pages 199–210, 1998.

[17] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *Text REtrieval Conference (TREC)*, pages 0–, 1994.

[18] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

[19] H. C. Wu, R. W. P. Luk, K.-F. Wong, and K.-L. Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Trans. Inf. Syst.*, 26(3), 2008.

[20] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 617–628. ACM, 2007.

[21] B. Zhao, C. X. Lin, B. Ding, and J. Han. Texplorer: keyword-based object search and exploration in multidimensional text databases. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 1709–1718. ACM, 2011.

[22] B. Zhou and J. Pei. Answering aggregate keyword queries on relational databases using minimal group-bys. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT 2009, Saint Petersburg, Russia, March 24-26, 2009*, pages 108–119. ACM, 2009.