THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRD3/11

BROAD: Diversified Keyword Search in Databases

Feng Zhao, Xiaolong Zhang, Anthony K. H. Tung and Gang Chen

March 2011

T e c h n i c a l R e p o r t

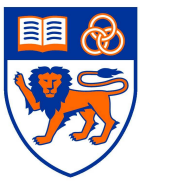
Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

BROAD: Diversified Keyword Search in Databases

Feng Zhao [#], *Xiaolong Zhang* [*], *Anthony K. H. Tung* [#], *Gang Chen* [*]
[#]*School of Computing, National University of Singapore, Singapore*
{*zhaofeng,atung*}*@comp.nus.edu.sg*
[*]*College of Computer Science, Zhejiang University, China*
{*xiaolongzhang,cg*}*@cs.zju.edu.cn*



NUS
National University of Singapore

School of Computing

NUS SOC Technical Report
Number TRD3/11
March 18, 2011

School of Computing
National University of Singapore
13 Computing Drive
Singapore 117417
Telephone: +65 6516 2727
Fax: +65 6779 4580
Homepage: http://www.comp.nus.edu.sg

BROAD: Diversified Keyword Search in Databases

Feng Zhao [#], Xiaolong Zhang [*], Anthony K. H. Tung [#], Gang Chen [*]
[#]School of Computing, National University of Singapore, Singapore
{zhaofeng,atung}@comp.nus.edu.sg
[*]College of Computer Science, Zhejiang University, China
{xiaolongzhang,cg}@cs.zju.edu.cn

National University of Singapore, School of Computing, TRD3/11
March 18, 2011

ABSTRACT

Keyword search in databases has received a lot of attention in the database community as it is an effective approach for querying a database without knowing its underlying schema. However, keyword search queries often return too many results. One standard solution is to rank results such that the "best" results appear first. Still, this approach can suffer from redundancy problem where many high ranking results are in fact coming from the same part of the database and results in other parts of the database are missed completely.

In this paper, we propose the BROAD system which allows users to perform diverse, hierarchical browsing on keyword search results. Our system partitions the answer trees in the keyword search results by selecting k diverse representatives from the trees, separating the answer trees into k groups based on their similarity to the representatives and then recursively applying the partitioning for each group. By constructing summarized result for the answer trees in each of the k groups, we provide a way for users to quickly locate the results that they desire.

Technically, our solution consists of three components. First, a new distance metric is used to capture both semantic and structural dissimilarity between answer trees. Second, based on this metric, we propose a tree-based algorithm to efficiently achieve result diversification. Finally, by coupling our partitioning solution with result summarization techniques, we allow users to decide which partition to drill down in order to obtain their intended answers. Extensive experiments were conducted and the results validate the feasibility and the efficiency of our system.

1. INTRODUCTION

With increasing amount of textual data being stored in relational databases, keyword search is well recognized as a convenient and effective approach to retrieve results without knowing the underlying schema or learning a query language [3, 16, 17, 14]. The result of keyword query is often modeled as a compact substructure, such as a tree or graph, which connects keyword tuples to include all the keywords. Potentially, a user could discover underlying relationships and the semantics based on structural answers.

However, keyword search queries can often return too many answers. This is because the semantics captured in a keyword query is limited, and the tuples that keywords are located in might come from different tables and connect with each other in many ways. As a result, exploring and understanding keyword search results can be time consuming and not user-friendly. To illustrate this, we describe a simple example on CiteSeerX[1] dataset. Figure 1 shows the schema graph G_S, in which nodes are associated with tables and edges indicate foreign key references.

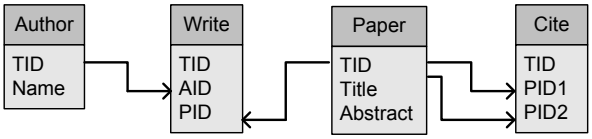


Figure 1: CiteSeerX Schema Graph

EXAMPLE 1. *Consider a keyword query on "skyline" and "rank" over the CiteSeerX dataset. There are* 78 *tuples containing the keyword "skyline", and* 729 *tuples containing the keyword "rank". A snapshot of keyword tuples are presented in Table 1, and part of the answers related to these tuples are shown in Figure 2. For clear illustration, we use "a" to denote an author and "p" to denote a paper. It can be seen that the relationship between them varies a lot even for fixed keyword tuples. Presenting and exploring the results of this keyword query will be difficult.*

A typical solution to large number of keyword search results is to return top-k answers according to relevant score [14]. Sophisticate ranking strategies have been developed to attempt to capture the search intention of a user. Without knowing the schema, however, it is hard for a user to explicitly express the preference. For instance, the query {skyline, rank} aims to discover the relationship between them, but it is difficult to

[1]http://citeseerx.ist.psu.edu/

Proceedings of the VLDB Endowment, Vol. X, No. Y

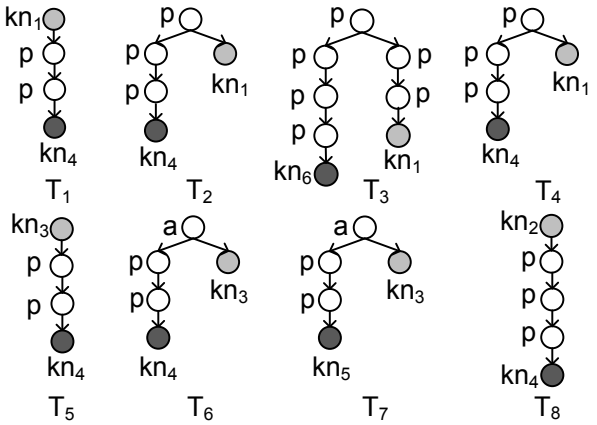


Figure 2: Search Result Examples

Table 1: The Snapshot of Keyword Tuples

ID	Content Excerpt
kn_1	The [Skyline] Operator
kn_2	[Skyline] with Presorting
kn_3	An Optimal and Progressive Algorithm for [Skyline] Queries
kn_4	Merging [Ranks] from Heterogeneous Internet Sources
kn_5	Why [Rank]-Based Allocation of Reproductive Trials is Best
kn_6	The PageRank Citation [Rank]ing

indicate which keyword is more important or what types of path connections are meaningful before a user realizes what can be found in the dataset. Even if it is possible to estimate users' preference, the top-k results usually include many overlapped answers that are redundant to present. This can be seen in Example 1 with an extreme case that T_2 and T_4 share two keyword nodes and even an identical answer structure.

Ideally, the results for keyword query would properly account for the interests of the overall user population [8]. In view of this, result diversification has been well studied in information retrieval community [8, 12, 4]. More explicitly, they try to put documents with broad information and different semantics in the first page of search interface. Consequently, the search engine improves users' satisfaction since each user has high possibility of efficiently finding interesting documents. Our aim here is to apply this idea to select diversified answer trees for keyword search over databases. For instance, we may choose T_1 and T_7 in Figure 2 since they represent different keyword tuples, and the connection structures are distinct as well. To make this possible, three new challenges must be overcome:

(1) Diversity Measurement: Intuitively, result diversification is a trade-off between having more relevant results of the "correct" intent and having diverse results in the top positions for a given query [12]. As such, aside from considering the relevance of the answers, we also need to take into account the pairwise difference between answers. Therefore, our first and the most important challenge is to define a meaningful measure between substructures tailored for keyword search in databases. Various efforts have been made to measure the dissimilarity of keyword search results [24, 10]. While we will discuss these papers in detail subsequently, it suffices to point out here that none of them capture both textual and structural information when trying to diversify keyword search answers.

(2) Query Answering: Result diversification is a NP hard problem in nature [12] and is thus expected to be computationally intensive. Although finding representatives in clustering problem is a candidate solution, it is imperative to notice that clustering method also has high computational cost. More importantly, the diversity quality of the clustering method is shown low compared with heuristic approaches [11]. Although we try to divide results into k groups, our objective is to make the distinction between k answers as large as possible.

(3) Result Representation: The ultimate goal of our paper is to facilitate better search experience and database usability. Since the original structural answers are complex and not easy to understand, we need to simplify them in order to let users quickly perceive the underlying difference between answers. To achieve this goal, the challenge is to effectively summarize the distinct features from many keyword search answers.

To overcome these challenges, we develop a novel system for browsing and diversified keyword searching in databases, i.e. BROAD (BROAD is an acronym for BROwsing And Diversified keyword searching). Our contributions towards diversified keyword search in databases are as follows:

- We have devised an effective kernel distance to measure the diversity of keyword search result. This metric integrates both the textual difference and the structural distinction in the answer trees.
- We have developed an efficient algorithm to find k diverse keyword query answers based on cover tree index structure. Unlike the post-processing approach, our solution seamlessly combines both relevant result discovery and diverse result set selection, allowing us to dynamically update the search result.
- We have provided a hierarchical browsing interface to further enhance our system. By coupling our solution with summarization techniques, we enable users to efficiently locate desired results by drilling down to relevant answers incrementally.
- We have conducted extensive experiments on two real datasets to show that our framework is both effective and efficient.

The rest of the paper is organized as follows. Section 2 defines the problem handled throughout this paper and proposes our new diversity measure. Section 3 introduces the BROAD system architecture. Section 4 presents the efficient index based solution, and the browsing interface of diversified result is described in Section 5. Our extensive experimental study is reported in Section 6. Section 7 reviews the related literature on keyword search in databases and result diversification. Section 8 concludes the paper.

2. PROBLEM DEFINITION

In this section, we introduce the keyword search modeling and describe the diversity problem studied in this paper. Furthermore, we propose a novel diversity measure to capture both content and structure information.

2.1 Keyword Search Modeling

We model our database as a graph in this paper. Database schema is a directed graph G_S called **schema graph**, in which nodes represent tables and edges represent foreign key references. For two tables R and S, the edge $R \rightarrow S$ indicates that the foreign key on S refers to the primary key on R. Note that there may exist multiple edges between R and S to represent multiple foreign key references. Given the schema graph G_S, the **data graph** G_D consists of nodes representing tuples and directed edges representing the foreign key references between tuples. Consider a l-keyword query q $\{c_1, c_2, \ldots, c_l\}$. Typically, the result of q on G_D is defined as follows.

DEFINITION 2.1. *(Answer tree)*
An answer tree T to the keyword query q is a rooted subtree of the data graph, satisfying: T contains all the keywords, and any subtree of T is not a valid answer tree. Denote the root of T as $n_r(T)$ and the node set of T as $N(T)$.

Generally, without restriction on the size of the answer tree, we will find a large number of meaningless answer trees due to long paths between nodes. Instead, we restrict the results to those trees that have a radius less than or equal to r. Note that the radius indicates the largest path length between the root node and leaf nodes. This is a common approach for keyword search in databases [17, 14].

DEFINITION 2.2. *(Res(q, r))*
Given keyword query q and radius r, the answer tree T is in the result set $Res(q, r)$ iff the path lengths between $n_r(T)$ and all the keyword nodes are less than or equal to r.

2.2 Diversity Problem Definition

We first assume that the dissimilarity between two answer trees can be measured by a distance function $dist(T_a, T_b)$ (with larger distance being more dissimilar). Based on this distance function, which will be discussed later in this section, we now formally state the problem of keyword search result diversification.

PROBLEM 1. *(Keyword Search Diversification)*
Given keyword query q and radius r, find a set of k answer trees $\mathcal{S} \in$ Res(q, r) which maximize $\min\{dist(T_a, T_b)\}$ where $T_a, T_b \in \mathcal{S}$.

Although there are various approaches to define diversity, we argue that the max-min diversification we adopted is suitable for keyword search in databases. Compared to the threshold based method [27], it avoids the difficulty of providing the query-dependent diversity threshold. Compared to the max-sum definition [12], we will find result set with lower variance, making it easier for user to distinguish between answer trees. Not surprisingly, the desired objective is NP-hard similar to the paper [12]. The proof follows a reduction from the k center problem and we omit it due to the space limitation.

2.3 Kernel Based Diversity Measure

The core of diversity problem is the need to measure the pairwise dissimilarity between answer trees, i.e. $dist(T_a, T_b)$. Here, we choose a kernel based method for this purpose and will explain our choice subsequently.

2.3.1 Answer Tree Kernel

Formally, a kernel function [22] is a function measuring the similarity of any pair of objects $\{x, x'\}$ in the input domain $\mathcal{X}$. It is written as $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$, in which ϕ is a mapping from $\mathcal{X}$ to a feature space $\mathcal{F}$. Given a set of examples $\{x_1, x_2, \ldots, x_m\}$, the Gram matrix is defined as the $m \times m$ matrix G^κ whose entries are $G^\kappa_{i,j} = \kappa(x_i, x_j)$. A kernel function is valid if and only if it is symmetric positive semidefinite, i.e. if any of its Gram matrices are symmetric positive semidefinite. Readers are referred to the book [22] for a comprehensive introduction on kernel methods.

To ensure efficient computation of the kernel, we utilize the subtree kernel [26] as the starting point since it is a linear complexity kernel for tree structural data. This kernel is extended from the state-of-the-art convolution kernel [13]. The basic idea is to express a kernel on a discrete object by a sum of kernels of their constituent parts. The features of the subtree kernel are proper subtrees of the input tree T. A proper subtree f_i comprises node n_i along with all of its descendants. Two proper subtrees are identical if and only if they have the same tree structure and the corresponding nodes are from the same table. Considering T_1 and T_7 in Example 1, all of their proper subtrees are shown in Figure 3. Both answer trees contain four different proper subtrees, and they share three of them, namely, f_1, f_2, f_3. The definition of subtree kernel is as follows.

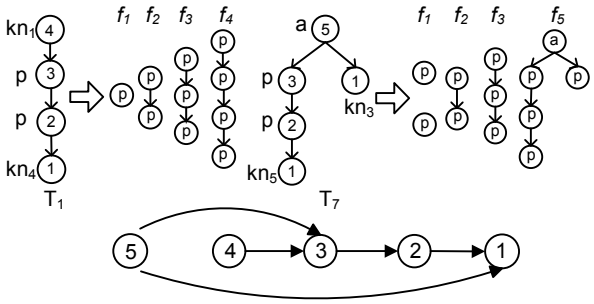


Figure 3: Kernel Example

DEFINITION 2.3. *(Subtree Kernel)*
Given two trees T_a and T_b, the Subtree Kernel is:

$$\kappa_S(T_a, T_b) = \sum_{n_a \in N(T_a)} \sum_{n_b \in N(T_b)} \Delta(n_a, n_b)$$

where $\Delta(n_a, n_b) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_a) I_i(n_b)$, and where $I_i(n)$ is an indicator function which determines whether the proper subtree f_i is rooted in node n.

Originally, the subtree kernel is designed to compare only tree structures without taking node contents into consideration. For example, the kernel score $\kappa_S(T_1, T_7) = 1 \times 2 + 1 \times 1 + 1 \times 1 = 4$ only because they share the substructures f_1, f_2, f_3. In our case, the comparison between answer trees need to consider node contents as well. Although the paper [7] has integrated textual information into the convolution kernel, their approaches are only suitable for parse tree in grammar analysis. Our paper is the first attempt to design a kernel for structural keyword search answers. The idea is to take $\Delta(n_1, n_2)$ as a fuzzy match between proper subtrees. Since answer trees contain textual information, we could compare the content similarity of two proper subtrees from two

answer trees that have the same structure. Let f_i^a be a proper subtree in T_a and f_i^b be a proper subtree in T_b that share the proper subtree f_i. We merge the textual content in the nodes of f_i^a and f_i^b into d_i^a and d_i^b and refer to them as documents. Next, we represent each document as $v = (w_1, w_2, \ldots, w_t)$ with each dimension corresponding to a separate term. If a term occurs in the document, its value in the vector is non-zero. Applying one of the best known schemes, i.e. TF-IDF weighting, we obtain $\kappa_D(d_i^a, d_i^b) = \langle v_i^a, v_i^b \rangle$ where v_i^a and v_i^b are the weighted term vectors of d_i^a and d_i^b respectively. Furthermore, the keyword query q provides another source of semantic information. Intuitively, f_i^a and f_i^b contribute more to the overall kernel if they share more keywords. Thus, we introduce a weight setting $w_{ab} = \sqrt{s/l}$ where s indicates the number of shared keywords and l represents the number of input keywords, yielding:

DEFINITION 2.4. *(Answer Tree Kernel)*
Given two trees T_a and T_b, the Answer Tree Kernel is:

$$\kappa_A(T_a, T_b) = \sum_{n_a \in N(T_a)} \sum_{n_b \in N(T_b)} w_{ab} \Delta'(n_a, n_b)$$

where $\Delta'(n_a, n_b) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_a) I_i(n_b) \kappa_D(d_i^a, d_i^b)$, and where $I_i(n)$ is an indicator function which determines whether the proper subtree f_i is rooted in node n.

The answer tree kernel serves as an effective method to map original answer trees to a kernel space. Next, we can define the answer tree kernel distance $dist(T_a, T_b)$ accordingly. However, in the original answer tree kernel, larger trees have higher chances to share many common features with any small tree. To overcome this drawback, we compute the normalized kernel, i.e.

$$\kappa(T_a, T_b) = \kappa_A(T_a, T_b) / \sqrt{\kappa_A(T_a, T_a) \cdot \kappa_A(T_b, T_b)} \quad (1)$$

Based on the normalized kernel, we can define the answer tree distance as $dist(T_a, T_b) = \sqrt{1 - \kappa(T_a, T_b)}$. One important property of answer tree kernel is shown in Lemma 2.1.

LEMMA 2.1. *Answer tree kernel is a valid kernel, and answer tree distance is a metric.*

PROOF. *For the convolution kernel, if the kernels on the subparts are positive semidefinite, the overall kernel is also positive semidefinite [13]. As the answer tree kernel accords with the convolution kernel format, we need to prove that $\kappa_D(d_i^a, d_i^b)$ is valid. This can be shown by the kernel definition because $\kappa_D(d_i^a, d_i^b)$ is computed explicitly in terms of a dot product. Therefore, the answer tree kernel $\kappa_A(T_a, T_b)$ is a valid kernel and we map the answer tree to one metric space. Consequently, we define a norm $||T|| = \sqrt{\kappa(T, T)}$ based on the normalized kernel, and then the L2 distance metric on this space is*

$$\begin{aligned} L2(T_a, T_b) &= ||T_a - T_b|| \\ &= \sqrt{\kappa(T_a, T_a) + \kappa(T_b, T_b) - 2\kappa(T_a, T_b)} \\ &= \sqrt{2(1 - \kappa(T_a, T_b))} \end{aligned}$$

The above deduction relies on $\kappa(T_a, T_a) = \kappa(T_b, T_b) = 1$ by substituting Equation 1. Finally, we conclude that $dist(T_a, T_b)$ is a metric since $L2(T_a, T_b) = \sqrt{2} dist(T_a, T_b)$. □

2.3.2 Alternative Methods

There exist several different ways to define the similarity between answer trees. We could extract a finite-length feature vector for each answer tree, and then map it to a feature space to calculate the similarity via dot product. However, explicitly defining an effective feature space needs domain expert knowledge. Another way is to adopt tree edit distance [9]. This metric is defined as the minimal number of edit operations to transform one tree to another. However, computing tree edit distance for trees T_a and T_b suffers an expensive computational complexity $O((|T_a| + |T_b|)^3)$ [9]. Besides, we can decompose answer trees into a set of nodes and utilize Jaccard's distance to measure the difference. This method is efficient but sacrifices the result quality. First, it only considers the exact match of nodes, but ignores the textual similarity between them. Second, it fails to measure the structural connections due to decomposition. Compared to these methods, our kernel based approach can be computed in linear time and capture both structural and textual similarity without the need for domain knowledge.

3. SYSTEM ARCHITECTURE

We next present the BROAD system architecture as in Figure 4. We try to use a pipelined framework to overcome the challenges we discussed earlier. When a user inputs one l-keyword query in the browsing interface, it will be sent to keyword search engine generating candidate answer tree set $\mathcal{T}$. Here we rely on the standard keyword search engine in graph databases, which discovers answer trees from the data graph building on top of relational databases [3, 17]. Note that this component can be easily replaced with the relational keyword search engine [16]. Our BROAD system builds the connection between user interface and keyword search engine. It mainly consists of three components: Cover Tree Indexer, Diverse Result Generator and Hierarchical Browsing Operator. The results from the search engine can be progressively inserted into cover tree index in an online fashion. Based on this index structure, we will discover diverse result set and interact with users in a hierarchical browsing manner. For better illustration, we briefly explain the functionality of these components in BROAD system as follows.

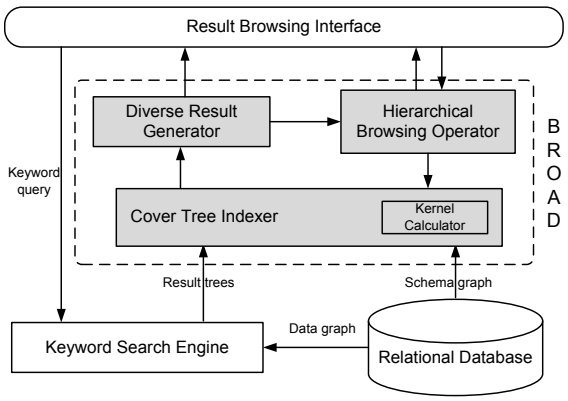


Figure 4: BROAD System Architecture

- Cover Tree Indexer: This module is the core of our system and will be discussed in details in Section 4. It dynamically manages the answer trees that are returned by

search engine. The kernel calculator serves as a subcomponent that computes the distance between answer trees based on the schema graph, so that the cover tree can index results effectively.

- Diverse Result Generator: The generator relies on the Cover Tree Indexer to discover k diverse results. This can not only directly show result to users, but also provides them with the Hierarchical Browsing Operator for further improvement.

- Hierarchical Browsing Operator: This component allows users to browse answer trees in a hierarchical fashion and will be discussed in Section 5. The hierarchy is constructed by partitioning answer trees into k groups based on their similarity to the k diverse results, and then recursively applying partitioning to each group. By summarizing the answer trees in each of the k groups, we provide a way for users to quickly locate the desired results.

4. METHODOLOGY

In this section, we propose an efficient algorithm computing the tree kernel distance. Based on this, we develop a cover tree based algorithm to solve the Problem 1. Alternative approaches are listed in Section 4.3.

4.1 Kernel Distance Computation

To compute the tree kernel distance, a naïve calculation follows naturally from the idea in Definition 2.4. Intuitively, this method checks all the possible combinations between nodes of two answer trees and sums up the shared parts to obtain the final score. It is straightforward but suffers from $O(|T_a||T_b|)$ computational complexity. Here we consider this problem from another aspect. The number of proper subtrees in a tree equals to the size of the tree. Let us consider Figure 3 again. T_1 has four proper subtrees $\{f_1, f_2, f_3, f_4\}$, and T_7 has five $\{f_1, f_1, f_2, f_3, f_3\}$. Therefore, we could directly enumerate all proper subtrees instead of checking every possible node combination.

Based on this intuition, we design a novel bottom-up algorithm to merge answer trees into a directed acyclic graph. The graph at the bottom of Figure 3 is generated from answer trees T_1 and T_7. The number inside each node represents the correspondence between a tree node and a graph node. For instance, the nodes with label 3 in two answer trees can be merged into the graph node with label 3. It is because they have the same structure f_3, in which all the nodes come from the "paper" table. Due to the bottom-up traversal, the children of newly accessed node must be mapped to certain graph node before it. Thus, by checking the child correspondences, we could easily determine whether this node should be mapped to existing node or we need to create a new graph node. At last, each graph node represents one kind of proper subtree, because we create a new graph node if and only if we discover a new proper subtree. In this example, the two answer trees are merged into the graph with five nodes, indicating that they contain five different substructures in total. Following Definition 2.4, we calculate and sum up kernel scores of all the substructures to derive the final kernel score.

The improved algorithm contains two major subcomponents as in Algorithm 1. Function **buildDAG** merges two answer trees into one directed acyclic graph G. Following the bottom-up order, we add nodes in T_a into the *leftset* of nodes in G and nodes in T_b into the *rightset* of nodes in G. We then utilize G in the **kernel** function. This component computes semantic scores based on the *rightset* and the *leftset* of each graph node, and adds them up to obtain kernel score. In the main algorithm, we need to derive the self kernels for T_a and T_b and the cross kernel between T_a and T_b. Finally, we can calculate the kernel distance $dist(T_a, T_b)$ in line 4. Concerning the computational cost, the merging part needs single bottom-up traverse of two answer trees, and the computing part has $O(|G|)$ complexity with $|G| \leq |T_a|+|T_b|$. Obviously, the total complexity of Algorithm 1 is linear to the answer tree size.

Algorithm 1: **KernelDistance**

Input: Answer trees T_a and T_b
Output: The kernel distance $dist(T_a, T_b)$
1 DAG $G_{ab} \longleftarrow$ buildDAG(T_a, T_b)
2 DAG $G_{aa} \longleftarrow$ buildDAG(T_a, T_a)
3 DAG $G_{bb} \longleftarrow$ buildDAG(T_b, T_b)

4 $dist(T_a, T_b)$=$\sqrt{1 - kernel(G_{ab})/\sqrt{kernel(G_{aa})kernel(G_{bb})}}$
buildDAG(T_a, T_b)
1 enqueue leaf nodes into queue Q and create DAG G
2 **while** *Q is not empty* **do**
3 dequeue node v from Q ; $found \longleftarrow false$
4 **foreach** *node $w \in G$ in reverse order* **do**
5 break if w and v have different heights, outdegrees, or provenances
6 **if** *w and v have the same children* **then**
7 **if** $v \in T_a$ **then** add v to $w.leftset$
8 **else if** $v \in T_b$ **then** add v to $w.rightset$
9 $found \longleftarrow true$; break
10 **if** *found = false* **then**
11 add a new node w to G
12 **if** $v \in T_a$ **then** add v to $w.leftset$
13 **else if** $v \in T_b$ **then** add v to $w.rightset$
14 add arcs in G from w to all children of v
15 **if** *$v \neq$ Root **and** parent(v)' children are processed* **then** enqueue node parent(v) into Q
kernel(G)
1 $\kappa_A \longleftarrow 0$
2 **foreach** *node $v \in G$ in bottom up order* **do**
3 $d_a \longleftarrow \bigcup$ text content of v.leftset
4 $d_b \longleftarrow \bigcup$ text content of v.rightset
5 $w_{ab} \longleftarrow \sqrt{s/l}$; $\Delta'(v) = \kappa_D(d_a, d_b)$
6 $\kappa_A += w_{ab}\Delta'(v)$

4.2 Cover Tree Based Diversification

We next describe a cover tree solution to find k diversity answers out of N answer trees. We assume $N > k$ throughout the paper, since it is trivial to return all the candidates as diverse results when $N \leq k$. The cover tree [6] is a metric tree to index data and perform nearest neighbor search in metric spaces. It is a leveled tree where each level is a "cover" for the level beneath it. Each level is indexed by an integer scale i which starts from zero (root node) and increases as we descend the tree. For instance, an cover tree in Figure 5 indexes fifteen results of Example 1. Every answer tree repeats in the

lower level after it first appears, so the lowest level contains all the answer trees. Assume that we use the cover tree CT to index our answer set $\mathcal{T}$ based on answer tree distances, and C_i to indicate answer trees in $\mathcal{T}$ associated with the nodes at level i. Cover tree obeys three important properties:

- *Nesting*: $C_i \subseteq C_{i+1}$
- *Covering*: For every tree $T_a \in C_{i+1}$, there is a tree T_b such that $dist(T_a, T_b) \leq 1/2^i$ and exactly one such T_b is a parent of T_a.
- *Separation*: For all trees $T_a, T_b \in C_i$, the distance from T_a to T_b is greater than $1/2^i$.

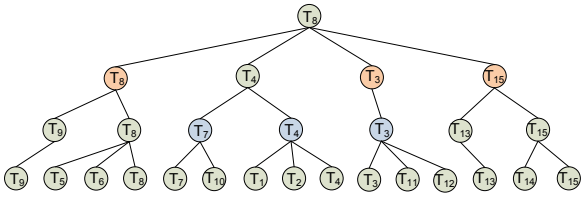


Figure 5: Cover Tree Example

The separation property of cover tree suggests that nodes at higher level are more diverse. Therefore, instead of discovering k diverse results from the whole answer set, we could make use of the cover tree to efficiently find good candidates for the result diversification problem. Unlike the original usage of cover tree, we propose a greedy algorithm to meet our need. Intuitively, the idea is to discover diverse results in the highest possible level over the cover tree. As illustrated in Algorithm 2, we access cover tree one level at a time, and then stop at the first level including at least k nodes, which is denoted as the working level C_i. If the size of C_i equals k, all the answer trees in this level are returned as diverse results. Otherwise, C_{i-1} is selected as the partial results, for that they are more separate in general according to the separation property. Next, we heuristically expand the farthest node in the working level until $|\mathcal{S}| = k$. When k is set to 3 for the cover tree in Figure 5, the algorithm proceeds as follows. In the level 0, it only contains the root node. Then the algorithm continues to the next level with four nodes. The number of nodes is larger than k, so this level becomes the working level and T_8 from the above level is selected as the partial result. Next, T_3 and T_{15} are further included by means of farthest expanding. Finally, we discover $\{T_8, T_3, T_{15}\}$ as diverse results for this running example. The cover tree index is constructed in $O(c^6 N \ln N)$ time for the expansion constant c [6]. The basic operation later in this algorithm is $setdist(T, \mathcal{S})$, i.e. $\frac{1}{|\mathcal{S}|}\sum_{T_a \in C_i} dist(T, T_a)$, which requires $|\mathcal{S}|$ distance computations. Since $|\mathcal{S}| \leq k$, we obtains its complexity as $O(k)$. This operation is performed $O(k|C_i|)$ times in the while loop, so the complexity of this algorithm is $O(k|C_i| \times k) = O(c^6 N \ln N + k^2 N)$ in terms of distance computations.

Furthermore, we propose an update method to support progressively updating the k diverse results in the BROAD system. The underlying idea is to check whether this newly added answer tree T_{new} affects the working level, and then adjust the k diverse results by means of swapping between T_{new} and T_{old} in original results. The $swapcost(T_{new}, T_{old})$ indicates the average distance change when we replace T_{old} with T_{new} in $\mathcal{S}$, i.e. $dist(T_{new}, \mathcal{S}) - dist(T_{old}, \mathcal{S})$. The complexity of Algorithm 3 consists of two components. The first is the beginning insertion with a complexity of $O(\ln N)$ [6]. The following part has $O(k^2)$ complexity since the $swapcost$ operation is performed $O(k)$ times. Thus, the total complexity of the update algorithm is $O(\ln N + k^2)$ in terms of distance computations.

Algorithm 2: **CoverTreeDiversification**

Input: Answer tree set $\mathcal{T}$, k
Output: The k diverse result set $\mathcal{S}$

Build cover tree CT from answer tree set $\mathcal{T}$
`// find the working level`
$C_{i-1} \longleftarrow NULL$
$C_i \longleftarrow C_0$
while $|C_i| < k$ **do**
  $C_{i-1} = C_i$
  $C_i = C_{i+1}$
`// discover` k `diverse results`
if $|C_i| = k$ **then**
  $\mathcal{S} \longleftarrow \bigcup$ all the answer trees in C_i
else
  $\mathcal{S} \longleftarrow \bigcup$ all the answer trees in C_{i-1}
  while $|\mathcal{S}| < k$ **do**
    find answer tree $T \in C_i \setminus \mathcal{S}$, s.t. $setdist(T, \mathcal{S}) = \max\{setdist(T, \mathcal{S}) : T \in C_i \setminus \mathcal{S}\}$
    $\mathcal{S} \longleftarrow \mathcal{S} \cup \{T\}$

Algorithm 3: **Update**

Input: Cover tree CT, Answer tree T_{new}, Result set $\mathcal{S}$
Output: The refined k diverse result set $\mathcal{S}'$

insert T_{new} into CT
if *working level* $C_i \in CT$ *is not changed* **then**
  if $|C_{i-1}| = k$ **then**
    $\mathcal{S}' \longleftarrow \bigcup$ all the answer trees in C_{i-1}
    set the working level to be C_{i-1}
  else $\mathcal{S}' \longleftarrow \mathcal{S}$
else
  $maxcost \longleftarrow 0$; $swaptree \longleftarrow NULL$
  foreach *answer tree* $T_{old} \in \mathcal{S}$ **do**
    if $swapcost(T_{new}, T_{old}) > maxcost$ **then**
      $maxcost \longleftarrow swapcost(T_{new}, T_{old})$
      $swaptree \longleftarrow T_{old}$
  if $maxcost > 0$ **then**
    replace $swaptree$ in $\mathcal{S}$ with T_{new}
    $\mathcal{S}' \longleftarrow \mathcal{S}$

To sum up, the cover tree based approach has several advantages. First, instead of diversifying results in the whole answer set, we utilizes the separation property to reduce the number of distance computations. Furthermore, cover tree supports progressive insertion with minor effort. Finally, the tree-like structure makes it a great tool for hierarchical browsing, which will be further explained in Section 5.

4.3 Alternative Solutions

We propose two state-of-the-art alternative approaches to solve the diversification problem. One solution is adapted from the farthest expansion algorithm [11, 12]. It maintains two sets of trees: the answer tree set $\mathcal{T}$ and diverse result set $\mathcal{S}$. Initially, the size of $\mathcal{T}$ is n and the size of $\mathcal{S}$ is zero. The farthest answer trees are iteratively moved from $\mathcal{T}$ to $\mathcal{S}$ until $|\mathcal{T}| = N - k$ and $|\mathcal{S}| = k$, as shown in Algorithm 4. $setdist(T, \mathcal{S})$ in line 4 is the average distance between answer tree T and all answer trees in $\mathcal{S}$, i.e. $\frac{1}{|\mathcal{S}|}\sum_{T_a \in \mathcal{S}} dist(T, T_a)$. Although guarantees a 2-approximation to Problem 1's optimal solution [12], this algorithm has complexity $O(N^2)$ in terms of distance computations. One possible relaxation of the quadratic complexity is to randomly select the first result and expand to rest $k-1$ results. However, this method is sensitive to the first random result, which needs multiple restarts to get the stable performance.

Another approach is derived from the k-medoids algorithm, as presented in Algorithm 5. The idea is to cluster candidates into k groups and select medoids as k diverse results. The number of distance computations is $O(Ik(N-k)^2)$, where I is the number of iterations. This method suffers high computational cost. Furthermore, it also requires starting from multiple initial medoids to approach global optimal results. The detailed comparison among these algorithms will be shown in the experimental section.

Algorithm 4: **FarthestExpanding**

Input: Answer tree set $\mathcal{T}$, k
Output: The k diverse result set $\mathcal{S}$
1 find T_a, T_b, s.t. $dist(T_a, T_b) = \max\{dist(T_a, T_b) : T_a, T_b \in \mathcal{T}, T_a \neq T_b\}$
2 $\mathcal{S} \longleftarrow \{T_a, T_b\}$
3 **while** $|\mathcal{S}| < k$ **do**
4 find answer tree $T \in \mathcal{T} \setminus \mathcal{S}$, s.t. $setdist(T, \mathcal{S}) = \max\{setdist(T, \mathcal{S}) : T \in \mathcal{T} \setminus \mathcal{S}\}$
5 $\mathcal{S} \longleftarrow \mathcal{S} \cup \{T\}$

Algorithm 5: **Clustering**

Input: Answer tree set $\mathcal{T}$, k
Output: The k diverse result set $\mathcal{S}$
1 Randomly select initial medoids $S = \{T_1, \ldots, T_k\}$
2 **for** $i = 1$ *to* I **do**
3 **for** $T \in \mathcal{T} \setminus \mathcal{S}$ **do**
4 Associate T with the medoid T_j, s.t.$dist(T, T_j) = \min\{dist(T, T_j) : T_j \in \mathcal{S}\}$
5 **for** *cluster* $S_j : j \in 1 \rightarrow k$ **do**
6 Update corresponding medoids $T_j \longleftarrow T$, s.t. $setdist(T, S_j) = \min\{setdist(T, S_j) : T \in S_j\}$

5. RESULT REPRESENTATION

BROAD system provides an interactive representation of search results. We implement a hierarchical navigation approach with visual interface to support user exploration.

5.1 Hierarchical Browsing

Hierarchical browsing is an effective approach to interact with users and can be elegantly supported by the cover tree structure. We proceed as follows. First, we separate answer trees in the working level into k answer tree groups $\mathcal{G}$ based on their kernel similarities to the k diverse results. A user then select a subset $\mathcal{H}$ of interest. Second, we fetch all nodes in next level covered by $\mathcal{H}$, and treat them as nodes in a new working level. Thus, we can perform Algorithm 2 again to obtain a new set of diverse results. This procedure iteratively proceeds until reaches the intended answer tree/s. For instance, T_8, T_3 and T_{15} in Figure 5 are diverse results found previously. We first assign T_4 to T_3 due to the kernel similarity and these four answer trees form three groups. Assume that users are interested in the group $\{T_3, T_4\}$, so we drill down to next level with answer tree set $\{T_3, T_4, T_7\}$. They are directly selected as new diverse answers because the size of this level equals to three.

5.2 Visual Interface

To support hierarchical browsing, we develop a visual interface so that users can easily browse and select preferred answer trees. In general, we merge a group of answer trees into circular view derived from the Circos project [19]. Circos has been proven to be a scalable and flexible visualization in many real world applications and research literature. In the following, we take answer tree T_7 to show the process of mapping an answer tree into a circle. Figure 6a depicts T_7 and it is transformed to the red part in Figure 6b and 6c. The root node and keyword nodes are mapped to segments, and pathes between nodes are mapped to ribbons. Answer tree contents, which will be discussed later, are selected as representative words around the circle. We also support the focused view when a user chooses certain answer tree. It is displayed with color and path structures, while other answer trees become transparent. T_7 in Figure 6c is shown in red and highlighted with the structure "author→paper" between the root node and the "skyline" keyword node. With the answer tree embedding, we presents the summarized view for a group of answer trees, which salvages space compared to the original layout. Furthermore, we utilize different colors to distinguish answer trees so that users can quickly capture how many answer trees are covered in the group. Also, users may refer to focused view for the detail of individual answer tree. This compact view is suitable for keyword search for that k is usually much less than one hundred, otherwise it is impossible to perceive search results.

Aside from structural summarization, representative words $\mathcal{W}_r$ are attached to the related segments in order to distinguish the circles that are on the same level of the hierarchy. Given any segment s, let the node it represents be n. The ribbons that connect s to other segments in the circle actually represent paths in the answer trees that connect n to other nodes in the answer trees. For each segment, candidate words $\mathcal{W}_c$ are selected from these pathes. Candidates for the highlighted root segment in Figure 6c are all the words from nodes in T_7. We then obtain $\mathcal{W}_r$ from these candidates as in Algorithm 6. In short, we compute a TF-IDF like score for candidate words, and select top candidates as representative words. As such, we sketch out the distinct contents of answer trees. The number of representative words selected depends on the width of the segment. For the green root segment in Figure 6f, the words "network", "distributed", "peer" and "neighbor" are selected

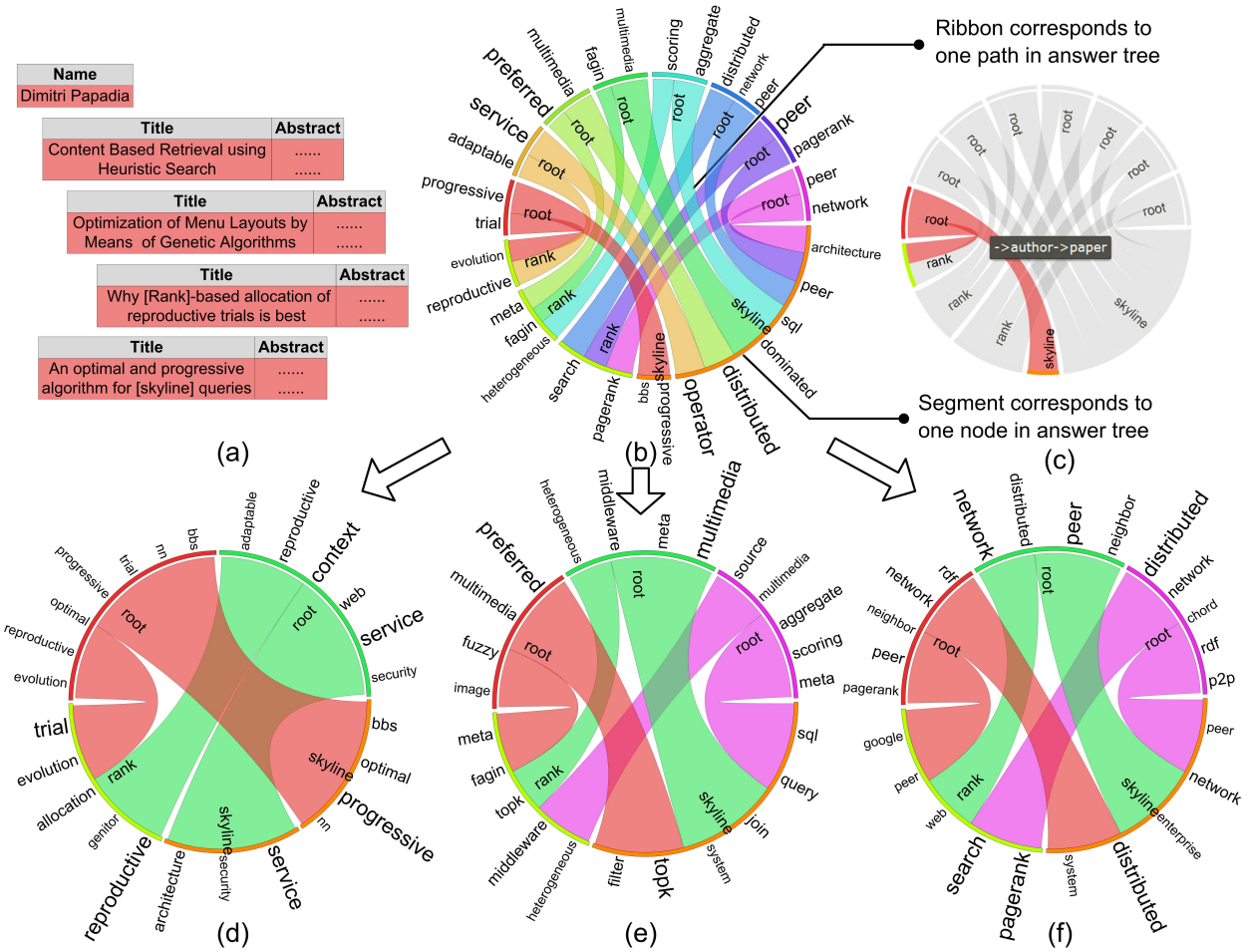


Figure 6: Result Representation

Algorithm 6: **Word Selection**

Input: Segment s, Answer tree groups $\mathcal{G}$
Output: Representative words $\mathcal{W}_r$

Obtain candidate words $\mathcal{W}_c$ of segment s
Obtain group g that segment s belongs to
foreach *candidate word* $w \in \mathcal{W}_c$ **do**
  $\mathcal{T}' \longleftarrow$ all the answer trees in g containing w
  $\mathcal{T} \longleftarrow$ all the answer trees in g
  $w.CoverRatio \longleftarrow |\mathcal{T}'|/|\mathcal{T}|$
  $\mathcal{G}' \longleftarrow$ all the groups in $\mathcal{G}$ containing w
  $w.Frequency \longleftarrow |\mathcal{G}'|/|\mathcal{G}|$
  $w.Score \longleftarrow$
  $w.CoverRatio \times \log(1/w.Frequency)$
Sort $\mathcal{W}_c$ with decreasing scores
Select $\mathcal{W}_r$ proportional to the width of the segment s

as representative words, since they have highest scores. To further emphasize the word distinctions within a segment, we present the selected words in different font sizes, according to their term frequencies in one segment. The words "network" and "peer" are highlighted with the biggest font size since their term frequencies are the largest in the segment.

The circular representation provides a summarized view both for structural and textual information about an answer tree group, which enhances the process of hierarchical browsing. In Figures 6d, 6e and 6f, we show three circles representing three groups of answer trees on the lower level of the browsing hierarchy. The left circle consists of two answer trees with one "rank" node and two "skyline" nodes. The content is mainly about the web services. The middle circle contains three answer trees. The major topic is the relationship between top-k query, skyline query and preference discovery. The right circle with three answer trees emphasizes the connection between skyline algorithm and distributed environment. In summary, circles can show distinct and summarized information about groups, which help users to browse and select desired answer trees. An interactive BROAD demo system is accessible over the Web[2].

6. EXPERIMENTS

We present experimental studies to evaluate the BROAD system in this section. Without loss of generality, we implemented the state-of-the-art graph based keyword search algorithm(*TopK*) [17]. To assess cover tree diversification (*DiverseK*), we compare it with *TopK*, Farthest Expanding algorithm(*FarthestK*) and Clustering algorithm(*ClusterK*).

Our experiments are executed based on the java code with 5 GB main memory. Table 2 explains the parameters used

[2]http://db128gb-b.ddns.comp.nus.edu.sg:8080/broad/

throughout this section. It also shows the range and the default values (in bold) of the parameters. In each experiment, we adjust one parameter while keeping the other one at its default value.

Table 2: Parameter Settings

Parameter	Description	Range
N	answer tree set size $\|\mathcal{T}\|$	$25, \mathbf{50}, 75, 100$
k	diverse result size	$2, 4, \mathbf{6}, 8, 10$

6.1 Datasets and Queries

We use two real datasets to assess our system. One is CiteSeerX, a collection of scientific and academic papers focusing on computer and information science. We choose this dataset for two reasons: i) It contains rich structural and textual information. This dataset maintains a large amount of paper abstracts as well as citations between papers; ii) It is a dataset for an online search engine associated with a query log. Another is Wikipedia, a well-known, web-based, collaborative encyclopedia. We test on a Wikipedia snapshot in March 2008[3]. Statistics about the graphs generated from these two datasets is shown in Table 3. Answer tree radiuses are set with respect to the closeness among nodes in different datasets.

Table 3: Dataset Statistics

Property	CiteSeerX	Wikipedia
Node count	$1,127,838$	$2,216,743$
Edge count	$3,414,540$	$76,797,970$
Graph size	~ 0.1 GB	~ 0.6 GB
Text size	~ 0.5 GB	~ 5.0 GB
Answer tree radius	6	3

To obtain a reasonable query set, we adopt a two-stage procedure. In the first stage, we extract meaningful query terms for each dataset. For CiteSeerX, we obtain a query log which is dominated by short queries with no more than 2 keywords ($> 94\%$). As such, we derive query terms from the log instead of directly using it. This is done by extracting frequent terms with term frequencies larger than 10. For Wikipedia, we extract ambiguous terms from disambiguation pages [1]. Ambiguous terms refer to more than one topics which Wikipedia covers. For example, "Healer" may refer to a film or a music album. We collect and use them as query terms. The second step is to generate keyword queries by randomly combining query terms. For query size l from 2 to 5, we produce 1000 initial queries for each value of l. In order to guarantee correctness, we test the queries using the keyword search engine, and filter queries that cannot produce enough answer trees. Then we rank the remaining queries according to the number of different keyword nodes in a descending order. Finally, we select the top-20 queries for each l, i.e. 80 queries per dataset.

6.2 Evaluation Metrics

In IR community, evaluating the accuracy of diverse query results is well studied and several evaluation metrics are established, such as S-recall and S-precision [28], α-NDCG [8], NDCG-IA [4] and so forth. The metrics extended from NDCG are not suitable for our problem, for these metrics rely on the result ranking. Therefore, we will evaluate our system based on S-Recall and S-Precision. However, we cannot directly make use of them, but need to carefully adapt them for keyword search in databases. Fundamentally speaking, most of these evaluation metrics are based on subtopics or nuggets, which provide semantics covered by answers. In the context of database keyword search, we are required to define the meaningful subtopics. For CiteSeerX dataset, we utilize each keyword node to represent a distinct subtopic inspired by the paper [10]. This is reasonable since keyword nodes include the most important meanings and different keyword nodes indicate different semantics. Following the idea in query generation, we extract the subtopics from disambiguation pages for Wikipedia dataset. Each keyword term is related to one subtopic shown in the associated disambiguation page. In this way, answer trees can be reliably mapped to subtopics. Let subtopics$_q$ for query q be the relevant subtopics in all answer trees, and subtopics(T) be the relevant subtopics in answer tree T. We formally define S-recall in database keyword search as follows:

DEFINITION 6.1. *(S-recall)*
Given k results for query q, the subtopic recall is:

$$\text{S-recall at } k = \frac{|\bigcup_{a=1}^{k} subtopics(T_a)|}{|subtopics_q|}$$

where $|subtopics_q| = |\bigcup_{a=1}^{N} subtopics(T_a)|$.

The above metric refers to the percentage of subtopics covered by one of the k results derived from the paper [28]. However, it is trivial to achieve recall of 100% by returning all answer trees in response to any query. Therefore we also define S-Precision as a complement to S-recall. The subtopics$_k$ refers to the ideal size of subtopics in k results, assuming that all keyword nodes contain distinct subtopics.

DEFINITION 6.2. *(S-precision)*
Given k results for query q, the subtopic precision is:

$$\text{S-precision at } k = \frac{|\bigcup_{a=1}^{k} subtopics(T_a)|}{|subtopics_k|}$$

where $|subtopics_k| = \sum_{a=1}^{k} |subtopics(T_a)|$.

Besides the subtopic definition, our S-precision still differs from the S-precision in paper [28]. Originally, S-precision is defined based on S-recall. Given S-recall r, S-precision equals to $minRes(\mathcal{S}_{opt}, r)/minRes(\mathcal{S}, r)$. $minRes(\mathcal{S}, r)$ indicates the minimal size of results having S-recall r. This definition is not straightforward in the first place, since it is derived from S-recall r instead of result size k. Moreover, the computation is impractical due to the hardness of generating the optimal solution. These two reasons drive us to alter the definition of S-precision. Nevertheless, our definitions of S-recall and S-precision are natural generalization of standard recall and precision measures.

Furthermore, in order to show the effectiveness of the kernel distance, we calculate the minimal kernel distance (Min-dist) used in Problem 1, i.e. $\min\{dist(T_a, T_b)\}, T_a, T_b \in \mathcal{S}$. Intuitively, if Min-dist is positively correlated with S-recall and S-precision, we can conclude that the objective of Problem 1 is reasonable and it indeed produces diverse results.

As for efficiency evaluation, we focus on the response time comparison for result diversification excluding the time to discover answer trees. This is because we use identical keyword

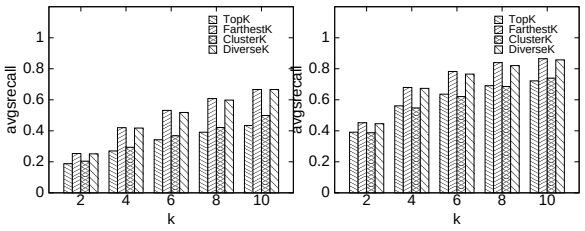
[3]http://www.archive.org/details/enwiki-20080312

search engine without affecting the comparison among algorithms. We also show the update cost for cover tree based solution to test its flexibility. In summary, by evaluating above metrics, we expect to investigate the overheads and gains of our framework.

6.3 Effectiveness Evaluation

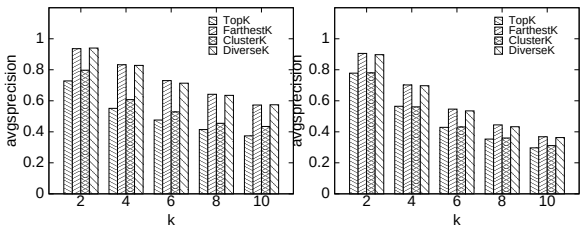
First, we vary k to study how this parameter affects the effectiveness of four approaches. To begin with, we present the average S-recall in Figure 7. It is clear that this metric has an ascending trend with the increase of k for all schemes. Since $subtopics_q$ remains the same for all schemes, S-recall is only dependant on $|\bigcup_{a=1}^{k} subtopics(T_a)|$. As k increases, this value increases as well, bringing about the ascent of S-recall. Nevertheless, we can easily notice their distinction in performance from the bar graph. $TopK$ performs worst because its ranking only relies on the relevance of the keyword query. Although $ClusterK$ groups answer trees according to the similarity, it is not optimized to select pairwise different medoids, so it also produces low quality results. For $FarthestK$ and $DiverseK$, it can be seen that they acquire the best results since they both apply the greedy strategy to discover results with respect to Problem 1's objective. Comparing the two datasets, the S-recall for Wikipedia in Figure 7b is higher than that of CiteSeerX in Figure 7a, since the number of $\mathcal{T}$'s subtopics in Wikipedia is smaller than that in CiteSeerX.

Figure 8 depicts the effect of k on the average S-precision for both datasets. The relative performances among all approaches are similar to the analysis for S-recall but the trend is negatively proportional to k. Our solution together with $FarthestK$ discovers the most diverse results among these four algorithms, since higher S-precision indicates smaller subtopic overlap between answer trees. Comparing Figure 8a and Figure 8b, we observe that the S-precision for Wikipedia is lower than that of CiteSeerX. This again is due to the limited number of subtopics in $\mathcal{T}$, which results in Wikipedia having a higher chance of obtaining overlapped result trees.



(a) CiteSeerX (b) Wikipedia

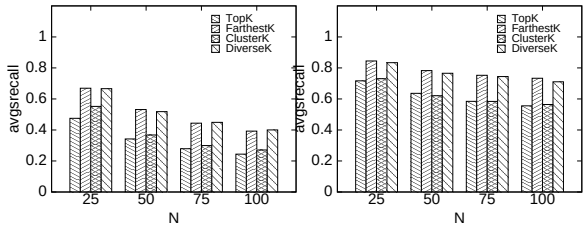
Figure 7: avg S-recall v.s. k



(a) CiteSeerX (b) Wikipedia

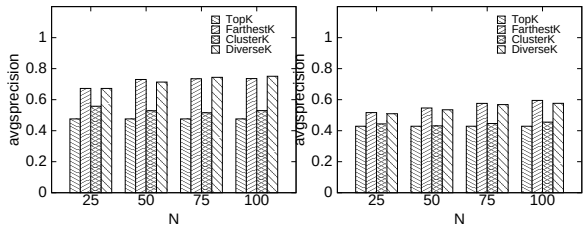
Figure 8: avg S-precision v.s. k

Next, we show the effectiveness measures by varying parameter N. The comparisons with respect to average S-recall and average S-precision are illustrated in Figure 9,10 respectively. Like the case of varying k, our solution as well as *FarthestK* gives the best quality answers and outperforms *ClusterK* and *TopK* by 20% to 50%. This result shows the effectiveness of our solution. In Figure 9, S-recall decreases as N increases. This is because $|subtopics_q|$ increases with N. When $subtopics_k$ remains the same with invariant k, S-precision is proportional to $|\bigcup_{a=1}^{k} subtopics(T_a)|$, i.e. the coverage of the k results. The results of *TopK* have the same coverage. The coverage of $clusterK$ has a small fluctuation, because the k medoids of clustering method are affected by the randomly selected initial medoids. For the other two algorithms, the quality increases with N. This observation is accordant with the intuition that we have more chances to discover better results as N increases. All these are reflected in Figure 10.



(a) CiteSeerX (b) Wikipedia

Figure 9: avg S-recall v.s. N



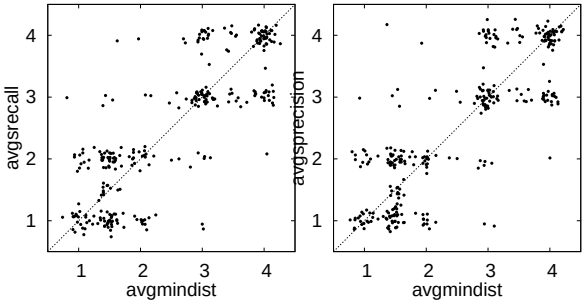
(a) CiteSeerX (b) Wikipedia

Figure 10: avg S-precision v.s. N

The last set of experiments focuses on correlation analysis between S-recall, S-precision and Min-dist. We observe that the absolute value of each metric varies with respect to different queries. Therefore, it is meaningless to directly compare the absolute value among queries. Instead, we transform them to the ranked value with each query. To illustrate, we calculate the ranking of the recalls $\{0.36,0.45,0.39,0.42\}$ for the four algorithms as $\{4,1,3,2\}$, in which smaller value indicates higher rank. If the related ranks of Min-dist are $\{4,1,2,3\}$, we obtain four points $\{(4\ 4),(1\ 1),(2\ 3),(3\ 2)\}$ in 2d figure for this query. As such, different query results are comparable. In Figure 11a, we display the scatter plot of S-recall against Min-dist for 80 queries on CiteSeerX. As the plot for Wikipedia dataset is similar, we won't display it. For better representation, we jitter the overlapped points in order to separate them. Generally speaking, this view indicates that there is a positive correlation since the majority of the points are located near the diagonal. The situation is similar in Figure 11b, which shows the relationship between S-precision and Min-dist.

Furthermore, we compute the correlation between these metrics using the Spearman's rank correlation coefficient [18]. It is a popular approach to examine correlation between the

ranked variables. The value has the range from -1 to $+1$ indicating the change from negative correlation to positive correlation. Denoting the coefficient between S-recall and Min-dist as ρ_{rm} and the coefficient between S-precision and Min-dist as ρ_{pm}, we calculate and obtain $\rho_{rm} = \rho_{pm} = 0.84$ in CiteSeerX dataset and $\rho_{rm} = \rho_{pm} = 0.68$ in Wikipedia dataset, which suggest that there is a strong ranking correlation between S-recall, S-precision and Min-dist. Therefore, we conclude that our distance measure significantly captures the diversity between answer trees.



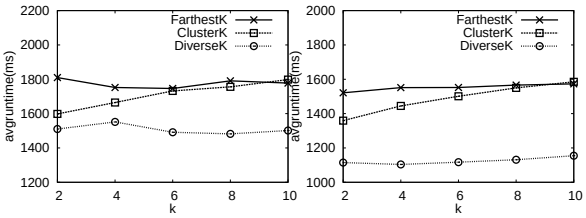
(a) (b)

Figure 11: Correlation Visualization on CiteSeerX

6.4 Efficiency Evaluation

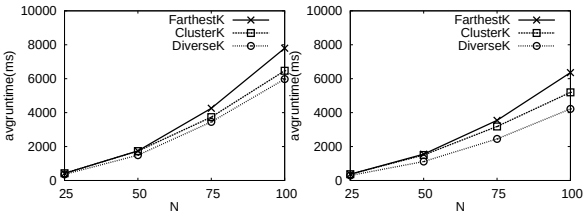
We next report the efficiency results. One important observation is the repeated distance computations waste a lot of time for all three algorithms. In our implementation, we cached the computed kernel distances in the main memory and reused them in case they appeared again. Because each distinct distance is calculated only once, the running time can be improved. Note that this trade-off between time and space is meaningful because the answer tree set size N is relatively small compared to the cardinality of the dataset. Figure 12 displays the relationship between running time and k. The computation times for *FarthestK* and *DiverseK* remain stable for different k. The former computes all pairs of kernel distances for the initial two farthest answer trees, so the remaining $k-2$ results can be discovered without computing any new kernel distance. In order to build the cover tree index, the latter already computes distances between the answer trees in the same level of the cover tree. Therefore, finding k results on certain working level merely relies on the cached distances. For the clustering method, however, the number of distance computations grows up as the increase of k. Its running time is smaller than *FarthestK*'s for a small k, but approaches to the running time of *FarthestK* when k equals to 10, suggesting that it has to compute all pairs of distances eventually.

Then the efficiency with respect to N is shown in Figure 13. Among three algorithms, *FarthestK* is the slowest, *ClusterK* ranks in the middle and *DiverseK* is the fastest. Because the average size of answer trees grows up, the average distance computation cost becomes larger. Therefore, the response time for cover tree based algorithm increases at a super-linear trend, but still saves about $15\% \sim 30\%$ against the *FarthestK*. Respecting to the running time for two datasets, we perceive that Wikipedia queries are faster than CiteSeerX queries on average. The underlying reason is the average answer trees size of Wikipedia is smaller than that of CiteSeerX. As a result, the individual distance computation takes less time for Wikipedia dataset. Besides, the cover tree based solution supports dynamical updates as shown in Algorithm 3. This operation averagely takes 47ms for CiteSeerX and 25ms for Wikipedia on the default parameter settings, which just incurs a small overhead compared to discovering from scratch.



(a) CiteSeerX (b) Wikipedia

Figure 12: avg Runtime v.s. k



(a) CiteSeerX (b) Wikipedia

Figure 13: avg Runtime v.s. N

In summary, we show that the BROAD system is both effective and efficient from the above experiments. The proposed kernel distance nicely captures the diversity of the answers. Aiming at the same objective, the cover tree based solution is comparable to the *FurthestK* algorithm, which is much better than the clustering method and the original top-k answers. Also, it has the best response time and can dynamically update k diverse results instantly.

7. RELATED WORK

Keyword search in databases has been extensively studied. Current approaches can be classified into two categories: schema-based ones and graph-based ones. The schema-based methods [16] generated join expressions based on database schema and produced the resulting tuple trees through SQL queries. The graph-based approaches [3, 17] materialized the database as a graph in which each node corresponds to a tuple. They discovered compacted substructures based on heuristic graph search. Many of recent works [15, 20] developed different ranking strategies in order to improve the search effectiveness. Our work here provides a new angle to the problem by introducing diversity as an important search criteria and hierarchical browsing as a natural way to find relevant answers.

The notion of diversity has been investigated in many different contexts. Search result diversification is a powerful approach to enhance user satisfaction in the IR community [8, 4, 12, 11]. They developed various diversity measures for documents, and effectively solved the diversity problem based on different diversification objectives. However, their diversity measures are designed for documents, so the approaches are not applicable to keyword search in databases with structural answer set. Apart from result diversification in information retrieval, database researchers studied this problem as well. Yu

et al. [27] introduced the notion of explanation-based diversity in recommendation systems. Vee et al. [25] diversified the query results by applying an inverted-list algorithm. Liu et al. [21] developed a feature selection algorithm in order to highlight the differences among structural XML data.

Two recent works considered diversification related to keyword search in databases. PerK [24] studied the personalized keyword search in databases considering the answer diversity. DivQ [10] solved a new problem with regard to discovering diversified schemas. Both of them focused on re-ranking the results aware of diversity. Differently, we develop an interactive system which allows users to perform diverse, hierarchical browsing on keyword search results. This approach is beneficial to users since they can quickly discover answers based on their interests.

A common approach for making large datasets tractable for interactive exploration is through a browseable hierarchy. Smith et al. [23] grouped and visualized the search results based on the rich categorizes. Abello et al. [2] described a node-link-based graph visualization that allows clustering and navigation of large graphs. Balzer et al. [5] developed the voronoi treemaps for the visualization of software metrics. To the best of our knowledge, this is the first work to provide an hierarchical visualization on the keyword search results. We utilize summarization technique to visualize a group of answer trees in a compact circular view. Users can better perceive contextual and structural information through the interactive interface.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced BROAD, a novel system that integrates the discovery of diverse results with the current keyword search engine in databases. Unlike previous works, we proposed a new kernel distance metric between answer trees which captures both the structural and semantic information. Moreover, a cover tree based approach was developed in order to quickly and progressively return diverse results. To further consolidate this interesting framework, we provided a hierarchical browsing approach that helps navigate users in refining and browsing keyword search results. The outcomes from an experimental study demonstrated that the BROAD system can provide broad views of the answers that are returned by keyword search engine.

As for future research, we'd like to generalize our BROAD system to other keyword search based applications such as keyword search on XML database, RDF database etc.. As a result, it will become a general system to diversify structural search results and help users to browse the answer sets from keyword search. However, integrating different types of search results with the unified kernel distance remains challenging. Furthermore, explaining the underlying meaning of each diverse result is definitely another interesting area for future work.

9. REFERENCES

[1] http://en.wikipedia.org/wiki/Wikipedia:Disambiguation.

[2] J. Abello, F. Van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, pages 669–676, 2006.

[3] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, P. Parag, and S. Sudarshan. Banks: Browsing and keyword searching in relational databases. In *VLDB*, page 1086, 2002.

[4] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, pages 5–14, 2009.

[5] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM symposium on Software Visualization*, pages 165–172, 2005.

[6] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, pages 97–104, 2006.

[7] S. Bloehdorn and A. Moschitti. Structure and semantics for expressive text kernels. In *CIKM*, pages 861–864, 2007.

[8] C. L. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR*, pages 659–666, 2008.

[9] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, pages 1–19, 2009.

[10] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. *DivQ*: diversification for keyword search over structured databases. In *SIGIR*, pages 331–338, 2010.

[11] M. Drosou and E. Pitoura. Comparing diversity heuristics. Technical report, Technical Report 2009-05. Computer Science Department, University of Ioannina, 2009.

[12] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

[13] D. Haussler. Convolution kernels on discrete structures. Technical report, University of California, Santa Cruz, 1999.

[14] H. He, H. Wang, J. Yang, and P. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, page 316, 2007.

[15] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[16] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, page 681, 2002.

[17] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, page 505, 2005.

[18] M. Kendall and J. Gibbons. *Rank correlation methods*. Edward Arnold. Oxford University Press, 1990.

[19] M. Krzywinski, J. Schein, I. Birol, J. Connors, R. Gascoyne, D. Horsman, S. J. Jones, and M. A. Marra. Circos: An information aesthetic for comparative genomics. *Genome Research*, pages 1639–1645, 2009.

[20] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.

[21] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. In *VLDB*, pages 313–324, 2009.

[22] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[23] G. Smith, M. Czerwinski, B. Meyers, G. Robertson, and D. Tan. FacetMap: A scalable search and browse visualization. pages 797–804, 2006.

[24] K. Stefanidis, M. Drosou, and E. Pitoura. PerK: personalized keyword search in relational databases through preferences. In *EDBT*, pages 585–596, 2010.

[25] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, pages 228–236, 2008.

[26] S. V. N. Vishwanathan and A. Smola. Fast kernels on strings and trees. In *NIPS*, 2002.

[27] C. Yu, L. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, pages 368–378, 2009.

[28] C. Zhai, W. Cohen, and J. Lafferty. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *SIGIR*, pages 10–17, 2003.