Templated Search over Relational Databases

Anastasios Zouzias
IBM Research - Zurich

Michail Vlachos[*]
IBM Research - Zurich

Vagelis Hristidis [†]
Comp. Science & Engineering
UC Riverside

ABSTRACT

Businesses and large organizations accumulate increasingly large amounts of customer interaction data. Analysis of such data holds great importance for tasks such as strategic planning and orchestration of sales/marketing campaigns. However, discovery and analysis over heterogeneous enterprise data can be challenging. Primary reasons for this are dispersed data repositories, requirements for schema knowledge, and difficulties in using complex user interfaces. As a solution to the above, we propose a TEmplated Search paradigm (TES) for exploring relational data that combines the advantages of keyword search interfaces with the expressive power of question-answering systems. The user starts typing a few keywords and TES proposes data exploration questions in real time. A key aspect of our approach is that the questions displayed are diverse to each other and optimally cover the space of possible questions for a given question-ranking framework. Efficient exact and provably approximate algorithms are presented. We show that the Templated Search paradigm renders the potentially complex underlying data sources intelligible and easily navigable. We support our claims with experimental results on real-world enterprise data.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Search and Retrieval

Keywords

keyword search, query recommendations

[*]The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 259569.

[†]Dr. Hristidis was partially supported by NSF grant IIS-1216007 and a Samsung GRO grant.

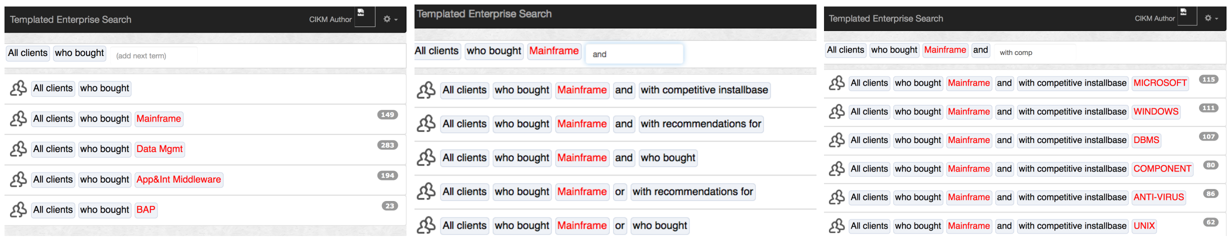
1. INTRODUCTION

Enterprises nowadays are sitting on billions of dollars' worth of data equity. Historical transaction data (e.g., products bought) and past interactional data with the client (e.g., responses to marketing campaigns) can offer valuable insights on the future buying preferences of the client. However, distilling useful information from these data repositories can be particularly challenging, especially for non-technical users. We can attribute this to three major factors:

- **Access Issues**: There exists a disparity between decision makers and data analysts that severely hampers direct access to data. Although strategy and business insights are driven by upper-level management, data access is serviced by the IT division of an enterprise. Therefore, gleaning the useful information may require several progressive rounds of requests between decision makers and data accessors. These multiple iterations reduce the efficiency of an organization, incurring unnecessary costs and time delays.
- **Data Issues**: Relating to the above, querying the data requires extensive knowledge of the underlying data and their schema to properly formulate the queries. Additional technical challenges that have to be addressed pertain to: (a) entity resolution problems (e.g., different client names of same entity per country) and (b) data scale.
- **Interface Issues**: Even when simple graphical interfaces do exist and offer data access to non-technical users, as the user interface progressively becomes more feature-rich, it will also become more difficult to operate. This induces a steep learning curve to adapt to new user interfaces. In addition, the various data tools cater for the needs of multiple groups of people with diverse access interests to the data. The end result is that the users only use a small fraction of an interface's functionality, but are exposed to its full complexity.

In this work, we propose a *guided* keyword search mechanism for retrieving information over relational databases[1]. Given a database schema, the proposed search mechanism receives as input keywords from the user such as ("*iphone*" or "*Boston*") and the system generates a list of queries that are valid on the given database schema, such as: "all clients who purchased an *iphone*", "all clients located in *Boston*", or similar queries corresponding to the keywords. Compound queries can be formed such as "all customers who bought *iPhone* and are located in *Boston* " if both keywords

[1]The proposed framework is applicable to any type of data source that consists of entities and relationships between them.


Templated Enterprise Search CIKM Author

All clients who bought (add next term)

All clients who bought
All clients who bought Mainframe 149
All clients who bought Data Mgmt 283
All clients who bought App&Int Middleware 194
All clients who bought BAP 23

Templated Enterprise Search

All clients who bought Mainframe and

All clients who bought Mainframe and with competitive installbase
All clients who bought Mainframe and with recommendations for
All clients who bought Mainframe and who bought
All clients who bought Mainframe or with recommendations for
All clients who bought Mainframe or who bought

Templated Enterprise Search CIKM Author

All clients who bought Mainframe and with comp

All clients who bought Mainframe and with competitive installbase MICROSOFT 115
All clients who bought Mainframe and with competitive installbase WINDOWS 111
All clients who bought Mainframe and with competitive installbase DBMS 107
All clients who bought Mainframe and with competitive installbase COMPONENT 80
All clients who bought Mainframe and with competitive installbase ANTI-VIRUS 86
All clients who bought Mainframe and with competitive installbase UNIX 62

Figure 1: Examples from the Templated Search interface. Potential queries are recommended on-the-fly. In red are shown the dynamic parts of the query templates. The system is also able to provide the counts of the result sets in real-time.

"iphone" and "Boston" are inputs. These natural-language queries should be generated by the system and also semantically map to valid SQL queries on the database.

End users want to receive answers to their questions without delving into details of the database schema, a requirement for issuing SQL queries, or learning how to use rigid and inflexible user interfaces [3]. Therefore, it is advantageous to put forward a new search paradigm that guides the user and eases the exploration and correlation over heterogeneous data sources.

In TEmplated Search or TES for short (Figures 1 and 7), queries are formulated using a simple textual description, allowing the user to ask queries like: "all clients who bought product X " or "all clients with expiring contracts for product Y" which we call *templates*. Queries consists of both *static* text that is predefined (i.e., customers, contracts, who, product, etc.) and *dynamic* text that is retrieved from the database. Moreover, the original query primitives (templates) can be compounded to form increasingly more complex query instances. We address the challenges that we identified above, in the following ways:

- Using the technology presented, the user does not need to issue SQL queries or even have (complete) knowledge of the underlying schema. The universe of potential queries is captured in templates, and can be augmented as the system matures.

- The system makes interactive recommendations about *valid* queries on the data, based on the templates created. The templates can be combined to articulate more elaborate query functionalities. The query recommendations are finally ranked so as to satisfy: *relevance*, *coverage* and *diversity*. Note that the keywords that the user inputs as part of the query need not necessarily exist in the same tuple or even in the same table of the database.

- To limit the interface complexity, we also use templates for the various visual components. The user is only presented with a simple interface that is relevant to the particular question. If the user asks for *"Contracts about Client X"*, then only a list of relevant contracts will be shown. In contrast, when the question is concerning *"Products bought by Client X"*, a different view will be displayed that only captures the relevant information.

Here, we focus on relational datasets, mainly because most mature enterprises store historical data in relational tables. However, the notions we present are directly applicable on any other data-storage infrastructure such as a graph database [31]. Our system is primarily targeted towards non-technical decision makers who would like to have *direct* query access to enterprise data.

Our work makes the following **contributions**:

1) We propose a tree-based query generation mechanism to articulate and validate the queries that best cover the user-input keywords (Section 2). Given a database schema we define a (rooted) tree whose node/edge information is dynamically populated based on the input query. Moreover, each rooted path of the tree models a valid query as a dynamically text-populated path. We propose an objective function (Equation (4)) so that valid queries/paths are further ranked to satisfy three criteria: *relevance*, *coverage* and *diversity*.

2) We present an $\mathcal{O}(n^2 \log(n) \log(k))$ algorithm for top k ranking of n paths that *optimally* solves the path (query) ranking problem under the proposed objective function (Section 3). We also introduce a scalable, $\mathcal{O}(nk^2)$ time, 2-approximate algorithm.

2. PROBLEM DEFINITION

2.1 Overview

Given a database schema that consists of a set of relation and their connections through primary-foreign keys, our goal is to search through the database with a simple keyword search. We propose a guided search mechanism that, given the user's keywords, generates a list of *valid* questions in natural language format, where by valid we mean that they correspond to syntactically correct SQL statements. One such example could be: "[All clients] [who bought] [product X]" (without the brackets), see Figure 1. TES will provide recommendations based on the keywords posed. The only assumption that we make about the user is that he/she has only a high-level knowledge on the entities that exist in the database (i.e., clients, contracts, products) and their relationships (i.e., clients bought products, etc).

An important design question is how one defines the query generation mechanism given a database schema. We propose a tree-based query generation mechanism (see Figure 3 for an illustrative part of the tree-based mechanism). Given a database schema, we define a rooted tree in which each node is associated either with static text (i.e., "All clients", "who bought" together with a list of synonyms) or with text that is dynamically populated from the database (i.e., "product X", "product Y"). Syntactically, each rooted path models a query text by concatenation of its node text, therefore all possible queries generated are defined as the union of all rooted paths of the tree. Note that the set of possible queries is *not* equal to the number of distinct paths, but grows with the number of entries in the database's relations. Finally, each rooted path semantically corresponds to a specific query on the given database.

For the remainder of the paper, we will assume the database schema in Figure 2 [2]. In this schema, there exist

[2]It is possible to use a graphical database as the underlying schema without altering any of the statements in this paper.

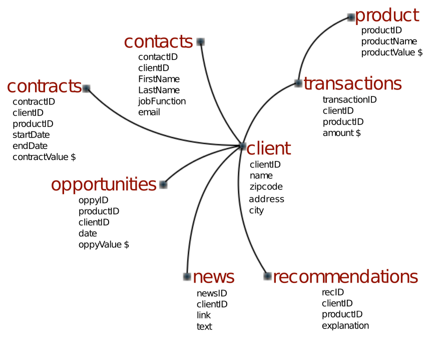


Figure 2: A simplified schema that we consider as our working example throughout the paper. Links correspond to primary-foreign key dependencies.

clients, *products*, *contracts* on products (e.g., maintenance), *opportunities* (a salesperson identified a product selling opportunity for a product) and *contacts* (people within the client organization with whom we interact, e.g., CEO, CMO, etc). Other potential tables/entities include: *news*, which correspond to recent news events about the client assembled from RSS feeds and Twitter, and *recommendations*, which stores the result of elaborate propensity models regarding the probability of a client to express interest in a particular product (consider, for example, recommendations as offered in Netflix). The schema also comes with additional information such as where each entity is stored, and also with potential synonyms for the entities and their attributes. For example, keywords such as *clients* and *customers* are synonyms and are mapped to the node (entity) `client`. Finally, all the text in all columns of interest in the database is indexed to support approximate or prefix search over all textual entities.

As input we also assume a set of *template queries* that we would like to answer on the system. For the above schema, a representative subset can be:

$\boxed{\text{Contracts of}}$ $\boxed{\text{client.name}}$

$\boxed{\text{Cxx Contacts of}}$ $\boxed{\text{client.name}}$

$\boxed{\text{All clients}}\boxed{\text{located in}}$ $\boxed{\text{client.address}}$ (or $\boxed{\text{client.city}}$)

$\boxed{\text{client.name}}\boxed{\text{who bought}}$ $\boxed{\text{product.productName}}$

$\boxed{\text{client.name}}\boxed{\text{with expiring contracts for}}$ $\boxed{\text{product.productName}}$

The templates consist of ***static nodes*** indicated in gray background color, and ***dynamic nodes***, shown in white color, which can be populated from a column of a table in the database. In the above examples, $\boxed{\text{Contracts of}}$ corresponds to a static node, whereas $\boxed{\text{client.name}}$ is a dynamic node. In essence, a query template encapsulates a *class* of potentially infinite queries. The templates are also accompanied by a valid parameterized SQL statement, that guides their evaluation on the database. Templates and their respective SQL statements are derived via analysis of the database query logs [29]. We do not further focus on this important research aspect, but direct the interested reader to related work [21, 37, 5].

While templates are few in number and answer simple queries, they may be combined, based on primary-foreign key relationships of the data entities, to formulate more complex queries. For example, the user can ask a question: *'All clients with expiring contracts for System P who bought SPSS'*, which combines two of the above templates. Templates as the above can be combined with `AND`, `OR` and `NOT` operators for formulating increasingly more complex statements.

Given the list of templates, the schema and the data, we need to generate valid query recommendations interactively and rank them accordingly. This is the focus of the upcoming sections.

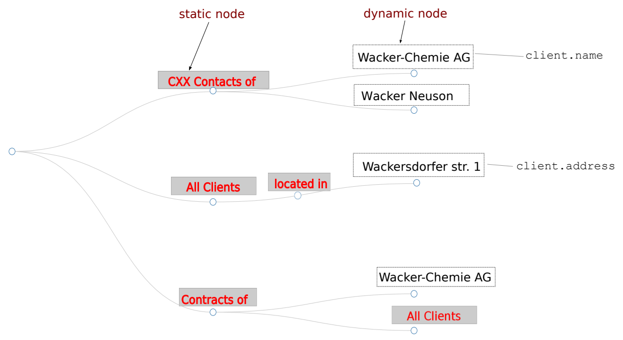


Figure 3: A simplified example of the tree-based query generation mechanism. For the sake of presentation, we demonstrates only 3 paths that have distinct return relations.

Example: Let's assume that the query "wacker" is posed (see Fig. 3). The system will try to determine whether this matches approximately some entry in the database, the static part of a query template, or both. The query matches two records at the attribute `name` of the entity `client`: 'Wacker-Chemie' and 'Wacker Neuson'. It also matches the address 'Wackersdorfer' of some `client` entity. Based on the above, an instance of the queries generated (query recommendations) is shown in Figure 3.

In what follows, we formally define the tree-based query generation mechanism (Section 2.2). It consists of two main steps:

(a) Given user's keywords, generate a list of "valid" question over the tree.

(b) Rank the valid questions/paths to satisfy three essential criteria: *relevance*, *coverage* and *diversity*.

2.2 Problem Definition

Now, we formally define the tree-based query generation mechanism, which we call Query Tree. We construct a rooted tree that consists of a set of "template paths". A minimalistic example of a Query Tree is shown in Figure 3.

Definition 1 (Query Tree). *Given a database schema, a Query tree (QT) is a rooted and directed tree $\mathcal{T} = (V, E)$ together with a labeling function $L : V \times \Sigma^* \mapsto (\Sigma^*, \Sigma^*, \ldots)$ where Σ is a fixed alphabet. QT has the following properties:*

1. *Root is "start" node.*
2. *Children of start are all relations of the schema.*
3. *Each relation may have as children (or descendants)*
   - *SELECT or PROJECT conditions*

- *`JOINS` to other relations, i.e., other relations connected through primary-foreign key; recursively these relations may have the same types of descendants*

4. *`AND`, `OR`, `NOT` are special connector nodes that combine children of relation nodes*

5. *Each $v \in V$ is a* static *or* dynamic *node. If v is a dynamic node and q is a list of keywords, then $L(v, q)$ is a ranked set of tuples from the relation that corresponds to v. Every textual result of $L(v, q)$ is ranked according to an IR-scoring function, e.g., [23]. If u is static, then $L(u, q) = L(u)$.*

6. *Dynamic nodes correspond to a ranked subset of tuples of a relation.*

In the above definition, the labeling function is required to frame the functionality of the dynamic nodes. For example, in Figure 3, q = 'Wacker' and let u be the dynamic node corresponding to *client.name*, then $L(\text{'Wacker'}, u) = \{\text{Wacker-Chemie AG, Wacker Neuson}, \ldots\}$. The top-level static nodes (children of start) define the type of the *return relation*, i.e., whether what will be returned is a client or a news element, etc. The lower-level nodes are in essence *condition relations* because they are used to constrain the results of the return relations. Each valid query generated may have one or more condition relations. This can be achieved using connector nodes such as `AND`, `OR` and `NOT`.

Example: Here, we explain the above terminology and demonstrate that each query contains all the information necessary for retrieving the semantically correct result set. Assume the query:
$\boxed{\text{All clients}}$ $\boxed{\text{in city}}$ $\boxed{\text{Berlin}}$ $\boxed{\text{and}}$ $\boxed{\text{with installbase}}$ $\boxed{\text{SPSS}}$.
The system will parse the query and *syntactically* identify the following: (1) return type is a list of clients (based on the $\boxed{\text{All clients}}$ static node); (2) client.city='Berlin'; (3) product.name='SPSS'; (4) fetch the client tuples that satisfy (2) and (3) as the connector is `AND`, i.e., clients with contracts for System P who also purchased SPSS.

Once the Query Tree is formed, we can use it to generate a set of relevant questions given a partial query from the user. The scope here is to provide variations on the current query, and also suggest ways of augmenting the query to form more complex and valid questions on the data.

Problem 1 (Questions Formulation). *Given a Query Tree and user keywords, create a list of valid questions (or paths).*

There are several challenges with Problem 1. Namely, there might be a match on schema and/or an instance or a tuple. For example, the keyword "clients" might match the static node "All clients" but might also match a client tuple with the name "Client Corporation".

2.3 Generation of Valid Questions (Paths)

Here, we discuss how we compute the set $\mathcal{P}$ of all valid questions for Q with respect to a Query Tree. When we refer to a *path* on the Query Tree, we imply that it is rooted at "start".

Definition 2 (Valid Path). *Given a Query Tree $\mathcal{T}$ and a query Q, a path (question) p is* valid *if (i) p (approximately) matches at least one keyword in Q, and (ii) p is minimal, that is, removing the last node of p would decrease the number of keywords matched in condition (i).*

First, in the above definition, condition (i) is loose so that a large number of paths is generated. Second, condition (ii) is used so that very long paths will not be included in the valid set of paths. The valid path generation procedure operates in two steps and prioritizes the search over the static text nodes first. In the first step, the procedure matches keywords from Q with only static text nodes. If there is an exact match with a keyword in this case, the keyword is removed; otherwise not. In the second step, the remaining keywords are used for populating all the dynamic nodes. Recall that for each dynamic node of the Query Tree, there is a corresponding attribute (or more) of some relation. Based on the text characteristics of any attribute (i.e., short company name, postal code, city, etc), we construct an appropriate text index over all tuples. Here, the search is based on a combination of approximate, prefix, edit distance and phonetic matching. In both steps, we store the information retrieval score between the query text and the node text for each node.

2.4 Main Primitive: Paths of Query Tree

We describe a few properties and characteristics of paths, because paths will be the core object of study in this paper. At the end of this subsection (Section 2.4.3), we propose a simple adjustment of the MMR diversification criterion to take into consideration a graph theoretic covering the property of paths, i.e., favor shorter paths which potentially contain relevant longer paths.

2.4.1 Metric Structure of Paths

Given two paths (questions) starting from the root node $p_1 = \text{root} \to t_1$ and $p_2 = \text{root} \to t_2$, our framework allows a restricted type of distance between p_1 and p_2. Namely, for a fixed tree $\mathcal{T}$ with arbitrarily positive weights on its edges, we define their distance as

$$\text{dist}(p_1, p_2) := \text{treeDist}(t_1, t_2), \tag{1}$$

where t_1 and t_2 are the final nodes of p_1 and p_2, respectively and also $\text{treeDist}(t_1, t_2)$ is the tree (or shortest path) distance[3] between node t_1 and t_2.

At this point, the above restriction on the tree metric might seem unnecessary. However, we will see later in Section 3 that such a restriction is crucial for the development of our algorithmic solutions.

2.4.2 Information Retrieval (IR) Path Scoring & Coverage

First, we define the IR score of a given node (dynamic or static). For a dynamic node v and a query Q, $L(v, Q)$ will return a ranked list of string corresponding to an attribute of some entity. The score of a (dynamic or static) node v, $\text{nodeScore}(v, Q)$ equals the information retrieval score returned in the valid question generation step.

Our ranking formula is an adaptation of the formula in [19]:

$$\text{IRScore}(p, Q) := \frac{\sum_{v \in p} \text{nodeScore}(L(v, Q), Q)}{\text{size}(p)} \tag{2}$$

[3]The shortest path distance between two nodes of a tree is defined as the sum of the edge weights over their connecting path.

where size(p) is the number of nodes in p and nodeScore(w, Q) is an appropriately chosen IR scoring function depending on its corresponding data source text characteristics; by default we use Okapi BM25 [23]. More precisely, if node v is a relation, then we define IRScore(v, Q) to be the maximum IR score of an attribute in the relation with respect to Q. If v is a select or project node, then we consider its text as a string (document) and compute the IR score as usual, i.e., BM25. Other ranking methods are also possible [35, 16]. Observe that the contribution of nodes that are far from the root node decreases linearly with their node distance.

Example: Figure 4 provides an example where Q is "Volvo clients bought SPSS". Here, three paths are displayed, most importantly the path $r \to v$, for which the score computation is expanded at the bottom panel of the figure. Another important characteristic of a path p that we would like to incorporate in our scoring function is the relevance of its descendants. The relevance of all descendants of a given partial path can be viewed as a *covering* property of a path. Namely, a path that has high relevance over its descendants should be more favorable to be included in the result list. For this reason, we defined the *coverage score* of a path p:

$$\text{CovScore}(p, Q) := \sum_{q \in \text{Desc}(p) \cup p} \frac{1}{\text{dist}(p, q) + 1} \text{IRScore}(q, Q). \tag{3}$$

where Desc(p) is the set of all descendant paths of p. The covering condition is an important property on paths as it guides the user towards more relevant paths.

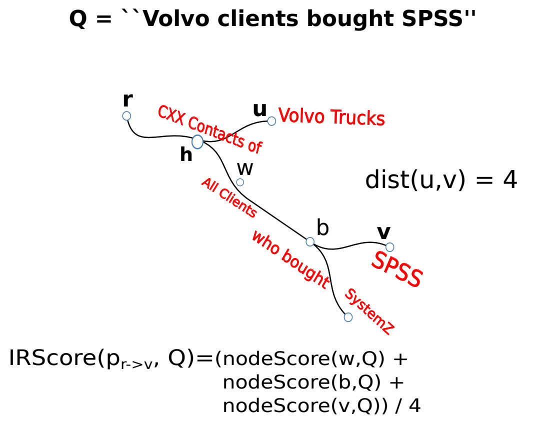


Figure 4: Information retrieval scoring and distance of paths. nodeScore(h, Q) is zero and therefore not included.

2.4.3 Putting Everything Together: Coverage, Diversity and Relevance

Now, we combine three aspects of any rooted path when one searches over the Query Tree: (i) *relevance*, the IR relevance of the path with respect to the query, (ii) *coverage*, the importance of all descendants of a sub-path (as the relevance of its descendants), and (iii) *diversity*, the dissimilarity or distance between two paths.

In most cases, one can expect the result set of valid potential questions to be huge. Therefore, a ranking criterion has to be decided. Returning the top-k valid questions with the highest score may not be the best approach, because they may be very similar to each other. So, it is important to incorporate a *diversity* factor.

Computing diverse and relevant results is a rapidly growing and active line of research in the database community [6, 17, 11, 14]. Most of the approaches are based on ranking documents/webpages given a query. Here, the main conceptual difference to prior work is that we diversify and rank *queries* instead of results.

In contrast to the diversity of web documents, where selecting a set of relevant and diverse results might be a satisfying solution [14], in our setting we must ensure that almost all questions can be (approximately) reached from the set of k questions presented, i.e., a *covering* condition must be satisfied by the selected paths. Past work has also examined how to incorporate topic coverage into a diversity function when the topics are organized as a hierarchy [38]; the intuition is similar to ours if we replace topics by queries, except that we compute the score based on the queries selected and not based on the results returned (which may be associated with several topics in [38]).

Search result diversification is a bi-criterion optimization problem in which one seeks to maximize the overall relevance of the document's ranking, while minimizing the redundancy on the documents returned. In general, the above bi-criterion problem is formulated as an intractable maximum coverage problem, and hence most algorithmic approaches are based on greedy solutions that provide only approximate solutions [17].

We adopt a variant of the well-accepted maximal marginal relevance (MMR) method of Carbonell and Goldstein [6] whose diversity and relevance balancing formula aims to identify k elements S (paths in our framework) that maximize:

$$\text{MMR-Path}(Q, S) = \min_{p \in S} \text{CovScore}(p, Q) + \lambda \cdot \min_{p, p' \in S} \text{dist}(p, p') \tag{4}$$

where CovScore(p, Q) is an arbitrary score of p given keywords Q; $|S| = k$, dist$(\cdot, \cdot)$ is a similarity measure between documents, and $\lambda > 0$ is a parameter specifying the tradeoff between relevance and dissimilarity within S. This bi-criteria objective maximizes the minimum relevance and dissimilarity of the chosen paths S (e.g., see Figure 5). Other diversity definitions have also been proposed, e.g., [12, 15]. The problem can be casted as follows:

PROBLEM 2 (QUESTIONS RANKING). *Given a set of valid questions/paths in the Query Tree and user keywords, compute k diversified questions/paths that maximize Equation (4).*

3. DIVERSIFIED QUESTIONS SELECTION PROBLEM

In the literature of document diversification, there is a well-understood connection between document diversification (using the maximal marginal relevance [6]) and dispersion problems on graphs [17]: diversification problems are typically reduced to a graph problem, known as dispersion problem [18, 24]). Here we follow the same approach, but with an essential difference to prior work: ***we diversify paths of trees***. The simplicity of tree structures, as opposed to general graphs, allows us to benefit in terms of both accuracy and efficiency. We present two provably accurate algorithms for Problem 2. Algorithm 1 is an $\mathcal{O}(n^2 \log(n) \log(k))$

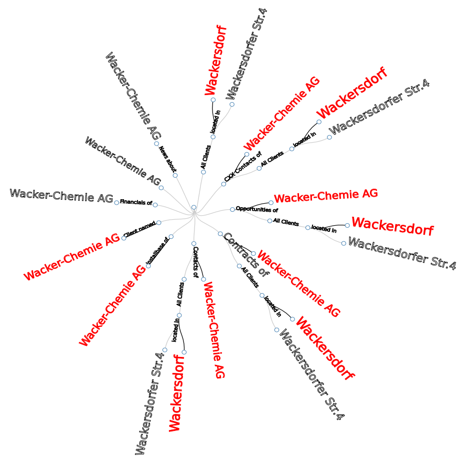


Figure 5: Input keywords are "wacker contracts" and k is set to 9. Font size is proportional to IR score for each node.

algorithm for top-k ranking of n paths that *optimally* solves the path-ranking problem. The algorithm discovers *optimal* solutions, but its quadratic dependence on the number of paths makes it prohibitive for our application; we seek for interactive response times on the order of milliseconds. To overcome this drawback of Algorithm 1, we also present a scalable, $\mathcal{O}(nk^2)$ time, 2-approximate algorithm (Algorithm 2) .

Maximal Minimum Dispersion Problem			
Graph Type	**Algorithm**	**Approx.**	**Time**
General	-	-	NP-hard
General	Greedy	2	$\mathcal{O}(n^2)$
Tree	Alg. 1,[8]	OPT	$\mathcal{O}(n^2\log(n)\log(k))$
Tree	Alg. 2	2	$\mathcal{O}(nk^2)$

Table 1: Summary of provably accurate algorithms for the maximal minimum dispersion problem on general graphs and trees.

In Table 1, we put our results into context. The last two rows correspond to our contributions. The first row shows that the problem is NP-hard in general graphs. The second row corresponds to the greedy 2-approximate algorithm for general graphs (note the quadratic dependency on n). The third row shows that the problem can be solved optimally on trees, whereas the last row demonstrates that computational savings can be achieved on the restricted case of trees. Therefore, the idea is to reformulate Problem 2 as an instance of the maximum minimum dispersion problem [18, 24] over trees.

First, we revisit the MaxMin dispersion problem, which is an intractable problem, i.e., NP-hard, in general graphs [7].

PROBLEM 3 (MAXMIN DISPERSION PROBLEM). *Given a positive integer k, a graph $G = (V, E)$ with $n = |V|$ and positive edge weights, compute a $S \subset V$ of size k that maximizes*

$$cost(S) := \min_{u \neq v \ | \ u,v \in S} dist(u, v) \qquad (5)$$

where $dist(u, v)$ is the shortest path distance on G.

Next we show how to reduce the objective value of Equation (4) to Equation (5). The reduction works by modifying the distance function $\text{dist}(\cdot,\cdot)$ to incorporate the scoring term of Equation (4) into the distance function. However, it is important to note that the results of Table 1 are in terms of *nodes* of a graph, they are not related to rooted paths as discussed here. Given the Query Tree restricted on the set of valid paths $\mathcal{P}$, we identify each node $v \in V$ with its corresponding path starting from the start node of the Query Tree and ending at v, i.e., start $\to v$. The following lemma makes formal the connection between Equation (4) on paths and the MaxMin dispersion problem on the underlying Query Tree.

LEMMA 1. *Let $\mathcal{P}$ be a set of valid paths over the Query Tree, Q be a query and k be an integer with $1 \leq k \leq |\mathcal{P}|$. Algorithm 1 computes a set of size k that optimally maximizes Equation (4) over all k-subsets of $\mathcal{P}$ in $\mathcal{O}(|\mathcal{P}|^2 \log(|\mathcal{P}|)\log(k))$.*

PROOF. For simplicity of notation, let $w(p_v)$ be equal to $\text{CovScore}(p_v, Q)$ as in Equation (3). First, observe that all paths have the same starting node, i.e., the start node of the Query Tree. Now, identify each path with its final node, i.e., the path $p_v :=$ start $\to v$ is identified with node v. Naturally extend this identification to set of paths. Given $\mathcal{P}$, let $T = (V, E)$ be the tree that contains the union of all nodes and edges contained in the set of paths $\mathcal{P}$. Moreover, for each $e = (u, v) \in E$, define $d_T(u, v) := \frac{1}{2}(w(p_u) + w(p_v)) + \lambda \cdot \text{dist}(p_u, p_v)$.

Now, note that for any $\mathcal{P}_S \subset \mathcal{P}$, it follows that

$$\begin{aligned}\min_{u,v\in S} d_T(u,v) &= \min_{u,v\in S}\left[\frac{1}{2}(w(p_u)+w(p_v)) + \lambda \text{dist}(p_u,p_v)\right] \\ &= \min_{p_u \in \mathcal{P}_S} \text{CovScore}(p_v, Q) + \lambda \min_{u,v\in\mathcal{P}_S} \text{dist}(p_u,p_v) \\ &= \text{MMR-Path}(Q, \mathcal{P}_S) \qquad (6)\end{aligned}$$

using the definition of d_T and $w(\cdot)$ in the first and second equality, respectively. The last equality follows by Equation (4). Now the `MaxMinDispersion` problem can be solved exactly on T in $O(|\mathcal{P}|^2 \log(|\mathcal{P}|)\log(k))$ time [8, Section 2]. Using Equation (6), the resulting nodes of the algorithm can be translated into a set of paths $\mathcal{P}_S$ of size k that maximize Equation (4) over all subsets of size k of $\mathcal{P}$. $\square$

Algorithm 1 solves the diversity ranking problem (Problem 2) optimally[4]. Algorithm 1 depends on the procedure `BoundedDisperse`, which solves a decision version of Problem 3, i.e., given $\mathcal{T}, d, k$ and λ, it returns S of size k such that $\text{cost}(S) \geq \lambda$. If such an S does not exist, it returns `failure`. Algorithm 1 uses Algorithm `BoundedDisperse` together with binary searching to compute the optimal objective value of Problem 3 with sufficient accuracy (Step 3), and then invokes Algorithm `BoundedDisperse` once more to return an optimal k-set S. Algorithm `BoundedDisperse` as presented here is a simplification of the implicit algorithm presented in [8, Section 2].

3.1 Scalable Greedy Diversification on Trees

Now we show that the time complexity of the greedy algorithm (second row of Table 1) for solving Problem 3 on trees can be significantly improved. The main observation is that during the execution of the greedy algorithm only the distances between the facilities and all other nodes are required.

[4]Recall that the problem is NP-hard for general graphs.

Algorithm 1 MaxMin Dispersion Algorithm on Trees

```
1: procedure MaxMinTreeDisp($\mathcal{T}$, $k$)                ▷ A tree $\mathcal{T}$, $k > 0$
2:     Let $\delta_{\min}$ and $\delta_{\max}$ be the smallest and largest weight in $\mathcal{T}$, resp.
3:     Binary search on $[\delta_{\min}, \delta_{\max}]$ using BoundedDisperse and approx. optimal value val of Equation (5)
4:     Return the $k$ set using BoundedDisperse($\mathcal{T}$, k, val)
5: end procedure
6: procedure BoundedDisperse($\mathcal{T}$, $k$, $\lambda$)     ▷ A tree $\mathcal{T}$, $k > 0$, $\lambda$: min. distance
7:     Let $C(s,t)$ be a cluster as Figure 6
8:     Add a facility to each $u_j$ s.t. $\text{dist}(u_j, s) > \lambda$; decrease $k$ accordingly
9:     Let $u_1, \ldots, u_l$ be leaves of $C(s,t)$ so that $d_1 \geq \ldots \geq d_l$
10:    Keep the maximum set of leaves so that their pairwise distances $\geq \lambda$ (always contain $u_1$ on the set)
11:    if $d_1 + \text{dist}(s,t) \geq \lambda$ then
12:        Add facility at $u_1$; decrease $k$ by one and remove node $u_1$ along with its edge
13:    end if
14:    Remove node $s$ along with all its adjacent edges
15:    Connect $u_i$ to $t$ with weight $\text{dist}(u_i, t)$
16:    Recursively call BoundedDisperse($\mathcal{T}, k, \lambda$)
17:    if fewer than $k$ facilities assigned then
18:        Report failure
19:    end if
20:    Return set of $k$ nodes that have assigned facility
21: end procedure
```

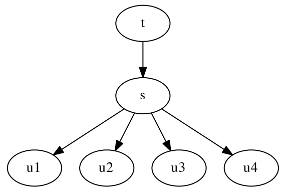


Figure 6: A cluster $C(s,t)$ of a 4-node tree (line 8 of Alg. 1)

Moreover, the distance between any fixed node and all other nodes in a tree can be computed by a breadth-first search (BFS) in $\mathcal{O}(n)$ time, see `updateMetaData` in Algorithm 2.

LEMMA 2. *Given a tree $\mathcal{T} = (V,E)$ with $n = |V|$, an integer $k > 1$, Algorithm 2 returns a 2-approximate solution to the maximal minimum dispersion problem on $\mathcal{T}$ using $\mathcal{O}(nk^2)$ operations.*

PROOF. The `UpdateMetaData` procedure is a breadth-first search which computes the distance between s and every other node in $\mathcal{T}$. Hence, `UpdateMetaData` requires $\mathcal{O}(n + |E|) = \mathcal{O}(n)$ time. Step 4 requires $\mathcal{O}(nk)$ time to compute the minimizer. In total, the algorithm requires $\mathcal{O}(nk^2)$ operations. As the algorithm is an instantiation of the greedy algorithm, its output solution is a 2-approximation [30, Theorem 2]. □

REMARK 1. *The running time of the greedy algorithm on general graphs is $\Omega(n^2)$ if only the edge weights are given, because an all-pairs shortest path computation is required to compute all node pairwise distances.*

4. EXPERIMENTS

For the experiments, we use real enterprise data from our host institution conforming to an expanded version of the schema in Fig. 2. The data cover a holistic view of clients including transactional and interactional data for a particular geography of our enterprise. To support approximate

Algorithm 2 Fast Greedy MaxMinDispersion on Trees

```
1: procedure FastGreedy($\mathcal{T} = (V,E)$, $k$)▷ A tree $\mathcal{T}$, integer $k > 1$
2:     Set $S = v$ for arbitrary $v \in V$ (preferably leaf).
3:     for $l = 1, 2, \ldots, k-1$ do
4:         Place the next facility on $V \setminus S$: $s_l := \arg\min_{v \in V \setminus S} \text{dist}(v, S)$.
5:         Call updateMetaData($\mathcal{T}$, $s$)          ▷ BFS starting on $s$
6:     end for
7:     Return set $S$
8: end procedure
9: procedure updateMetaData($\mathcal{T} = (V,E)$, s) ▷ Each node $v \in V$ maintains $\text{dist}(v,s)$ for every $s \in S$
10:    Create queue $Q$ and enqueue $s$ in $Q$; label $s$ as visited
11:    while $Q != \emptyset$ do
12:        $u \leftarrow Q.pop()$
13:        for Edges $e = \{u, v\}$ in adjacentEdges(s) do
14:            if $v$ is not visited then
15:                Label $u$ as visited
16:                $Q = Q \cup v$; $\text{dist}(s,v) = \text{dist}(s,u) + \text{dist}(u,v)$
17:            end if
18:        end for
19:    end while
20: end procedure
```

search over all textual entries, text is indexed using Apache Lucene/SOLR [2]. Search over the dynamic nodes of the Query Tree is based on a combined weighted sum of approximate, prefix, edit distance and phonetic match. Our test system contains a total of approximately 16 million indexed entities. The index size of the textual terms was on the order of 20.2 GBytes. In Figure 7 we show snapshots of the Graphical Interface of the system.

Path Scoring and Selection: First, we qualitatively demonstrate the effectiveness of our framework using a few representative examples. Given a set of keywords and a positive integer k, we perform the following steps: First, using the input keywords, we generate the set of all valid paths. Then, we use Algorithm 1 to highlight k paths that maximize the objective function defined in Section 2.4.3. Figure 8 demonstrates the generation of the Query tree and the paths selected for a progressively formed query search. Figure 5 depicts an actual query on the system with keywords "wacker contracts" and $k = 9$, and also shows that (i) a diversified set of paths is selected, and (ii) dynamic nodes `client.name` `client.address` and `client.city` have been populated.

Evaluation: Here, we present the experimental evaluation of the diversification algorithms we proposed in Section 3. We compare the efficiency in terms of running time for three diversification algorithms: `OPTtree` which is Algorithm 1, `FastGreedy`, which corresponds to Algorithm 2, and `Greedy`, which corresponds to a naive implementation of the greedy algorithm on trees, i.e., brute-force computation of all pairwise distances of the tree. Moreover, we evaluate Algorithm 1 in terms of the average path score (relevance) and average path distance (diversity).

Figure 9 depicts the running time of the algorithms versus the number of nodes selected for a tree with 1350 nodes, a maximum depth of 9 and 582 leaves. This figure suggests that both greedy solutions are superior to `OPTtree`. Moreover, Figure 9 demonstrates the quadratic dependency of the greedy algorithms for large values of k. Figure 10 depicts the running time of the algorithms for $k = 25$ and trees having size between 30 and 1350 nodes. This figure confirms the scalability of Algorithm 2 for constant values of k.

Parameter setting: Figure 11 depicts the average shortest path distance over k selected paths using Algorithm 2, see

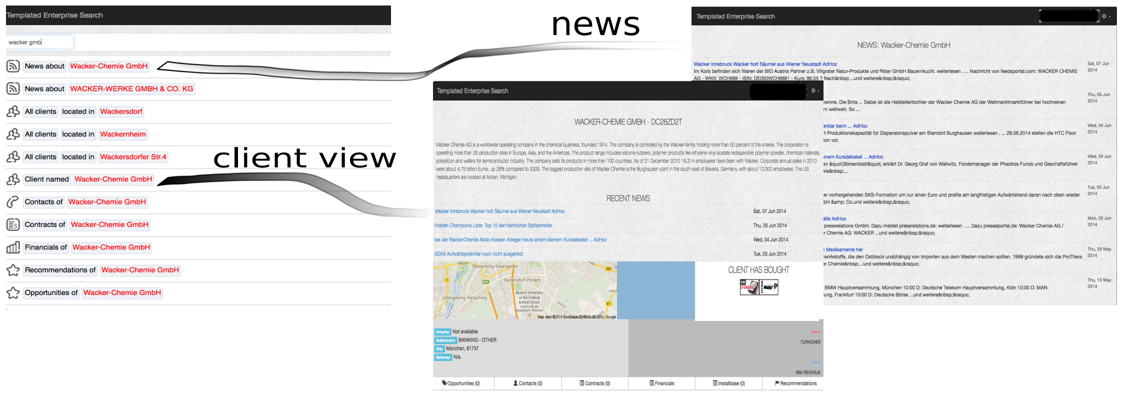
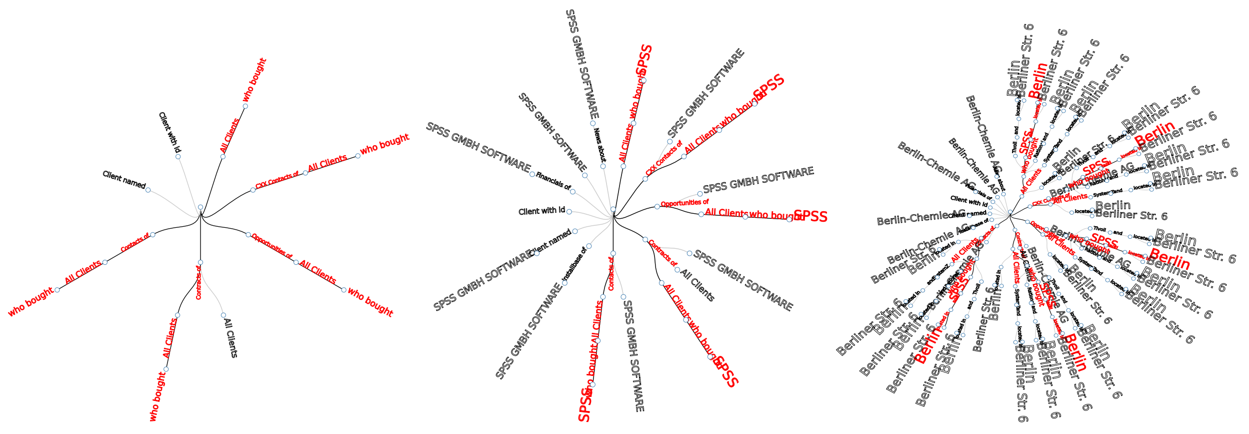

news
client view
Templated Enterprise Search
wacker gmb
News about Wacker-Chemie GmbH
News about WACKER-WERKE GMBH & CO. KG
All clients located in Wackersdorf
All clients located in Wackernheim
All clients located in Wackersdorfer Str.4
Client named Wacker-Chemie GmbH
Contacts of Wacker-Chemie GmbH
Contracts of Wacker-Chemie GmbH
Financials of Wacker-Chemie GmbH
Recommendations of Wacker-Chemie GmbH
Opportunities of Wacker-Chemie GmbH
NEWS: Wacker-Chemie GmbH
WACKER-CHEMIE GMBH - DC28Z2T
RECENT NEWS
CLIENT HAS BOUGHT

Figure 7: Left: An instance of the Query Interface generating valid queries based on the query templates. Right: The Answer Interface directs to different view templates to best accommodate the user's keywords.



(a) Query: "clients bought". (b) Query: "clients bought SPSS". (c) Query: "clients bought SPSS Berlin".

Figure 8: Progressive query path generation. In red are the queries selected that offer the best relevance, coverage and diversification as in Equation (4).

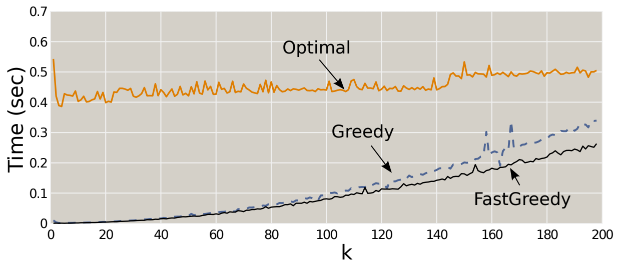


Figure 9: Elapsed time versus number of paths selected (k) for a generated tree with 1350 nodes, 582 leaves, and maximum depth of 9.

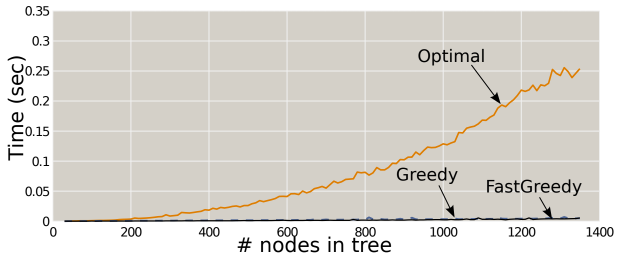


Figure 10: Elapsed time versus size of tree. We generate a tree with 30 to 1350 nodes and step 10.

Equation (1). It is evident that as k increases the average distance decreases after peaking out. This peak happens at the same value of $k \approx 20$ for all values of λ depicted. This suggests that a good value of k for our system is approximately 20. A similar experiment can be used to tune one's system before deployment.

Now, we turn our attention on how to set the parameter λ, which controls the relative importance between relevance and diversity. Figure 12 depicts the average path score over the k paths selected using Algorithm 2. As anticipated, the average relevance score of the paths selected eventually decreases as k increases. For fixed small values of k, we observe that the average relevance score is sensitive to the values of λ. We suggest fixing $\lambda \approx 0.6$ so that a fair trade-off between diversity and relevance exists.

User study: Even though our platform is intended for non-technical users, we also wanted to evaluate how effective the system can be for technical users. We gave 15 questions in plain English that can be answered by our system to 5 users, familiar with the database schema. Questions ranged from simple to elaborate. An indicative sample of questions that

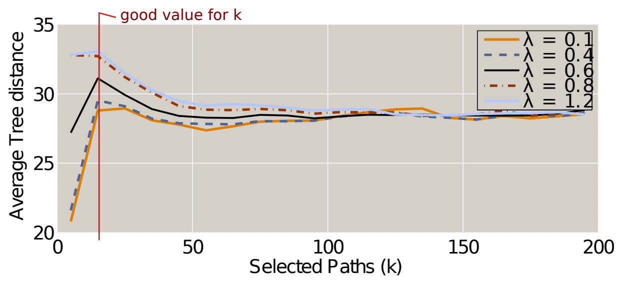


Figure 11: Average tree distance of paths for various values of k and λ.

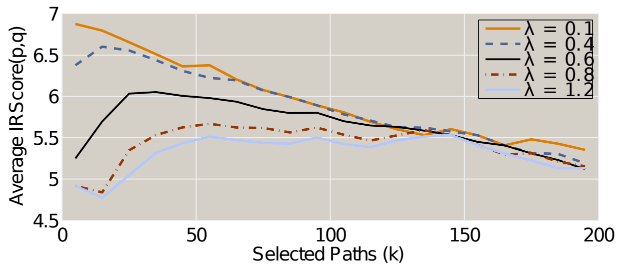


Figure 12: Relevance of paths (IRScore(p, Q)) for various values of k and λ.

were given included the following:

- ”All clients who bought [product name]”
- ”All clients with recommendations for [product name] who have not bought [product name]”
- ”All clients who bought [product name 1] and [product name 2]”
- ”News about [client name]”
- ”Cxx contacts of [client name]”
- ”All clients who bought [product name 1] and with recommendations for [product name 2]”
- ”All clients in industry [industry name] who bought [product name] and with expiring contracts in the next 6 months for product [product name]”

...and so on.

We asked them to write the query for each question in SQL and also to use our system, after a brief introduction of the Templated Search interface. We measured the time taken by each user to formulate the question, when issuing the query in SQL, or when using the Templated Search interface. Fig. 13 shows boxplots of the time taken by the users to formulate the questions. It is apparent, that even for advanced database users the presented system can offer distinct advantages. Users also commented that they found the new search paradigm to be very powerful and at the same time simple to use, and particularly useful for answering questions pertinent to the operations of our enterprise.

5. RELATED WORK

Keyword search on databases: DISCOVER [20], DBXplorer [1], BANKS [4], and others, have proposed ways to search structured databases using keywords. The key idea is that, given a set of keywords, the system looks for trees of tuples connected through primary-foreign key links (*candidate networks*), that collectively contain all the query keywords. Other ranking methods have also been proposed: (i) **Aggregation and OLAP**: IBM’s SQAK [34] (SQL Aggregates with Keywords) allows users to compute a variety of aggregate queries.Given such a keyword query and the schema graph, the system will compute candidate aggregate SQL queries and return the ones with highest score to the user. IBM’s Keyword-Driven Analytical Processing (KDAP) [36] combines intuitive keyword-based search with the power of aggregation in OLAP. (ii) **Semantic Web**: There are also works on using keyword search on semantic graphs (e.g., RDF graphs) [25]. SemSearch [25] assumes each keyword of the query is either a class (e.g., person), an instance (e.g., John) or a property (e.g., has-job-title). (iii) **Web search**: Given that Web search engines have an increasing number of vertical structured databases, such as tickets or products, there has also been work on finding the right database and attributes to match a web keyword query [33]. The above works do not consider progressive building of structured queries in an autocomplete-like fashion, nor distinguish between return and match relations. Moreover, query (candidate networks) semantics in keyword search on databases [20, 1] are different as each relation in the query must contribute (or link to) a query keyword. Instead, our queries (valid paths) are generated based on a combination of the templates and the query keywords, e.g., for a single-keyword query we may have a multi-relation question, but DISCOVER or DBXplorer would only have single-relation queries (the same holds for the results in BANKS). Also, query diversity is not considered. Finally, in our model the return relation is a primary class citizen, which allows our queries to be converted directly to natural language questions.

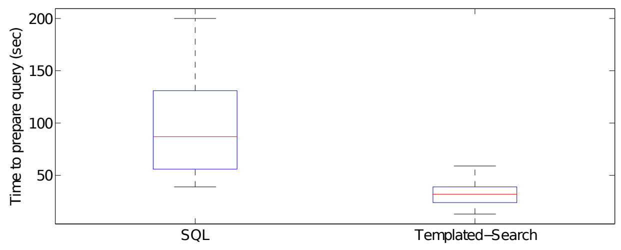


Figure 13: Boxplots of time of formulate proper search query in SQL and using the Templated-Search.

Database query construction interface: In [37], given an initial keyword query, they construct a query interpretation tree, each node corresponds to a decision – i.e., a question to the user. A sequence of such questions are asked until a query is constructed. In [13], this work is extended to leverage ontologies to group candidate queries, so fewer questions are posed to the user. The above works, neither support an autocomplete-style query construction, nor consider the diversity among the proposed queries.

Query forms: Another approach is to predefine a set of query forms (templates) and then let the user select a form and instantiate it. In [21] techniques are presented to automatically generate a set of forms for a relational database, using the data properties, without using a query workload. In [22] the authors show how to modify existing forms to create new forms. QURSED [27] is a tool to create query forms for XML data. In [9] keyword search on databases is combined with a form-based search. Limitations of using forms include: users rarely like to fill out forms; the number of forms to cover all possible queries is too high; no diversity measures are supported.

Natural language interfaces to databases: Much research was conducted especially in the 70's and 80's, on automatically converting natural language to SQL (e.g., [28]).

Facebook graph search: This is the project closest to our work. Users start with a few keywords and the system suggests possible search queries on the social graph. The Unicorn [10] system is the underlying index of the Facebook Social Graph on top of which Graph Search operates [32]. Although we do not know the details of how queries are generated, we speculate that the method has been developed specifically for social search and may not be easily adapted to other schemas, e.g., enterprise search. Also, there is no mention of how (if at all) diversity and coverage are addressed [26].

6. LIMITATIONS AND CONCLUSION

We have introduced a scalable, templated search technique over relational databases. Our approach guides the users and lets them easily query the potentially complex underlying data schema. The technical contributions of our work include:

- A tree-based question generation framework that is based on information retrieval and graph theoretical tools.

- Two provably accurate algorithms for path diversification over rooted trees: an optimal and a scalable 2-approximation algorithm.

Speaking in SQL lingo our system supports the following: multi-way joins, selections, projections, unions, intersections and exclusions over sets, and simple orderings of the results. Current *limitations* include: the absence of 'Group By' and aggregate queries operations. 'Group by' queries could easily be detected and expressed with the incorporation of static nodes using keywords such as 'per' or 'every'. Finally, the ideas in [34] could be applied in the future to support aggregate queries. In the long term, we plan to include support for basic data-mining operations. For example, when the desired outcome of a query is not already present in the data, but is the result of some analytic operation (e.g., clustering or classification). This will broaden the scope of our solution, allowing it to answer even more complex tasks.

7. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *Proc. ICDE*, pages 5–16, 2002.

[2] ApacheFoundation. Lucene/solr. http://lucene.apache.org/solr, 2013.

[3] Z. Bao, B. Kimelfeld, and Y. Li. Automatic suggestion of query-rewrite rules for enterprise search. In *Proc. SIGIR*, pages 591–600, 2012.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. ICDE*, pages 431–440, 2002.

[5] S. Bhatia, D. Majumdar, and P. Mitra. Query suggestions in the absence of query logs. In *Proc. SIGIR*, pages 795–804, 2011.

[6] J. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proc. SIGIR*, pages 335–336, 1998.

[7] B. Chandra and M. M. Halldórsson. Approximation algorithms for dispersion problems. *J. Algorithms*, 38(2):438–465, Feb. 2001.

[8] R. Chandrasekaran and A. Daughety. Location on Tree Networks: P-Centre and n-Dispersion Problems. *Mathematics of Operations Research*, 6(1):50–57, 1981.

[9] E. Chu, A. Baid, X. Chai, A. Doan, and J. Naughton. Combining keyword search and forms for ad hoc querying of databases. In *Proc. SIGMOD*, pages 349–360, 2009.

[10] M. Curtiss, I. Becker, T. Bosman, S. Doroshenko, L. Grijincu, T. Jackson, S. Kunnatur, S. Lassen, P. Pronin, S. Sankar, et al. Unicorn: A system for searching the social graph. *Proc. VLDB*, 6(11):1150–1161, 2013.

[11] V. Dang and B. W. Croft. Term level search result diversification. In *Proc. SIGIR*, pages 603–612, 2013.

[12] V. Dang and W. B. Croft. Diversity by proportionality: An election-based approach to search result diversification. In *Proc. SIGIR*, pages 65–74, 2012.

[13] E. Demidova, X. Zhou, and W. Nejdl. Efficient query construction for large scale data. In *Proc. SIGIR*, pages 573–582, 2013.

[14] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Rec.*, 39(1):41–47, Sept. 2010.

[15] M. Drosou and E. Pitoura. DisC diversity: Result diversification based on dissimilarity and coverage. *Proc. VLDB*, 6(1):13–24, 2012.

[16] G. J. Fakas, Z. Cai, and N. Mamoulis. Size-l object summaries for relational keyword search. *Proc. VLDB*, 5(3):229–240, Nov. 2011.

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

[18] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21(3):133 – 137, 1997.

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

[20] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *Proc. VLDB*, pages 670–681, 2002.

[21] M. Jayapandian and H. V. Jagadish. Automated creation of a forms-based database query interface. In *Proc. VLDB*, pages 695–709, 2008.

[22] M. Jayapandian and H. V. Jagadish. Expressive query specification through form customization. In *Proc. EDBT*, pages 416–427, 2008.

[23] K. S. Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: development and comparative experiments - part 1. *Inf. Process. Manage.*, 36(6):779–808, 2000.

[24] B. Korte and D. Hausmann. An analysis of the greedy heuristic for independence systems. In P. H. B. Alspach and D. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 65 – 74. Elsevier, 1978.

[25] Y. Lei, V. S. Uren, and E. Motta. SemSearch: A Search Engine for the Semantic Web. In *Proc. EKAW*, pages 238–245, 2006.

[26] X. Li and M. Boucher. Under the hood: The natural language interface of graph search. *http://goo.gl/bPlHb*, 2013.

[27] Y. Papakonstantinou, M. Petropoulos, and V. Vassalos. QURSED: querying and reporting semistructured data. In *Proc. SIGMOD*, pages 192–203, 2002.

[28] A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proc. IUI*, pages 149–157, 2003.

[29] J. Pound, S. Paparizos, and P. Tsaparas. Facet discovery for structured web search: a query-log mining approach. In *Proc. SIGMOD*, pages 169–180, 2011.

[30] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):pp. 299–310, 1994.

[31] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases.* O'Reilly Media, Incorporated, 2013.

[32] S. Sankar. Under the hood: Indexing and ranking in graph search. *http://goo.gl/jHKCK*, March 14 2013.

[33] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured annotations of web queries. In *Proc. SIGMOD*, pages 771–782, 2010.

[34] S. Tata and G. M. Lohman. SQAK: doing more with keywords. In *Proc. SIGMOD*, pages 889–902, 2008.

[35] A. Termehchy and M. Winslett. Keyword search for data-centric xml collections with long text fields. In *Proc. EDBT*, pages 537–548, 2010.

[36] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *Proc. SIGMOD*, pages 617–628, 2007.

[37] X. Yang, C. M. Procopiuc, and D. Srivastava. Recommending join queries via query log analysis. In *Proc. ICDE*, pages 964–975, 2009.

[38] W. Zheng, H. Fang, and C. Yao. Exploiting concept hierarchy for result diversification. In *Proc. CIKM*, pages 1844–1848, 2012.