# Exploratory Keyword Search with Interactive Input

Zhifeng Bao [+] Yong Zeng [*] H.V. Jagadish [#] Tok Wang Ling [*]
[+]RMIT University, Australia [*]National University of Singapore [#]University of Michigan
zhifeng.bao@rmit.edu.au zengyong@nus.edu.sg jag@umich.edu lingtw@comp.nus.edu.sg

## ABSTRACT

Due to the intrinsic ambiguity of keyword queries, users usually need to reformulate their queries multiple times to get the desired information. Even worse, users either have no way to precisely specify their search intention, or have limited domain knowledge on the data to precisely express their search intention. Moreover, they may just have a general interest to explore the data by keyword query. Therefore, our goal is to design an exploratory search paradigm that is able to bring humans more actively into the search process, in order to meet various user information needs, ranging from simple lookup to learning and understanding of the data.

Besides, keyword queries against data with structure, such as XML, can run into multiple difficulties: how to identify the search target; more types of ambiguity arise as a keyword can be part of the structure as well as content of data, etc. Effectively addressing these requires solutions to multiple challenges. While some have been addressed to some extent individually, there is no previous effort to develop a comprehensive system to meet these important user needs and meet all of these challenges.

Therefore, we propose a framework called ClearMap [1] that natively supports visualized exploratory search paradigm on XML data. In particular, we offer an interactive and visualized mechanism to present the outcome of the query, enable user to explore and manipulate the underlying data to either quickly find desired information or learn the relationship among data items, as well as provide interactive suggestions when their expected results do not exist in the data. A preliminary version of ClearMap and its source code are available for try at http://xmlclearmap.comp.nus.edu.sg.

## 1. INTRODUCTION

The primary interaction between users and databases starts from the traditional structured query, which assumes that users are to some extent familiar with the content and structure of the database and also have a clear understanding of their information needs. As databases get larger and accessible to a more diverse and less technically oriented audience, new forms of data exploration become increasingly more attractive [3]. Since keyword search paradigm frees users from the hindrance of learning query syntax and database schema, it is widely adopted as an entry to building the exploratory search engine.

Regarding keyword search over databases, we also observed an evolution on the granularity of users' information needs when more and more (semi-)structured data become available for search. They are not merely content to simply retrieving the basic facts from database (like most users do in web search), but desire for advanced information needs, such as learning and understanding the results as well as the underlying data [6]. Such advanced needs are substantial in practice for at least two reasons: (1) Users do not have enough background knowledge on either the structure or the content of the underlying data; so a typical search cycle is actually a process of examining and comparing results and possibly reformulating queries until user's desired information needs are met [8]. (2) Users usually assume the thing they are looking for is out there, but users may be stuck if the desired information does not even exist in the database, regardless how they reformulate their queries.

Our goal is to design an exploratory search paradigm that is able to bring humans more actively into the search process, in order to meet various user information needs, ranging from simple lookup to learning and understanding of the data. Specifically, we aim to support human users' navigational browsing and exploration of semi-structured data, and ultimately they may not need to reformulate their query for resubmission many a time.

Doing so requires addressing challenges in multiple spheres, including index design, result retrieval and result visualization. So before we design our exploration model, it is worth having an overview of the challenges brought by keyword search over semi-structured data and the past endeavors made in addressing them. Compared to web search, search over semi-structured data brings extra challenges: (1) how to identify the search target of a user query (while in web search web page is the target); (2) more types of ambiguity may occur, because a keyword can be part of the structure as well as the content of data, and users may also have limited or wrong knowledge of the schema and value of the underlying database [7]; (3) semi-structured data is usually modeled as a tree or a graph, and the search result is a subtree or a subgraph, and two issues remain open problems: (a) how to present the results in a human-interpretable way and convey the relationship between data items [4]; (b) how to find subtrees (or subgraphs) of appropriate size, including relevant yet non-overwhelming information [2].

Over the years, we have made continuous efforts to address every individual challenge above. E.g. we tried to identify and rank the search targets and constraints to address the keyword ambiguity [2], proposed visualization methods to display results at different granularity [13], and provided suggestions (and explanations deriving such suggestions) to cater for the mismatch between user's information needs and the returned results [12]. But without inter-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*SIGMOD'15,* May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2735361.

action with user, these challenges cannot be addressed thoroughly [4]. Taking search target identification as an example, even though [2] tried to find the promising search targets for a single query, we may not be able to guarantee that the target found fits to each individual user's search context. More importantly, those individual works employ individual indexing and retrieval methods to work. Therefore, it is demanding to build a comprehensive search engine that meets the above challenges systematically, in terms of index design, result retrieval and result visualization.

To achieve our goal, we build a keyword search engine called ClearMap [1], which helps users get beyond finding to understanding and learning of the information resources in the database, and ClearMap is comprehensive in that it meets all of the above challenges. In particular, we make the following contributions.

- We define a center search concept, called Visualized Search Object (VSO), around which both structure and content of the database can be indexed and organized by search engine, and results in form of VSO can be manipulated by users.
- We utilize the inter-connectivity of the data in semi-structured databases to design a novel exploration model, to allow users to expand their manipulation from results to the whole database.
- We natively support the exploratory search paradigm with interaction and visualization enabled by making a seamless integration of VSO and our previous experiences [13, 2, 12].
- We design a multi-level index on VSO which can maximize the reuse of index information by different components, as well as providing support to visualization of huge subtree.

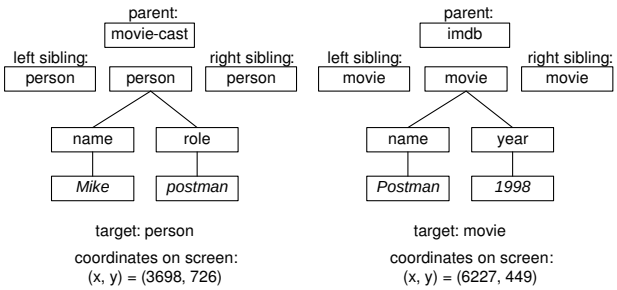
As a result, ClearMap has the following central characteristics:

- Interactive - It allows users to use the visual output as a visual interface for specifying additional operations on either the underlying data or the search results.
- Exploratory - For users who do not find their desired information by their initial query, they can explore the visualized data directly without reformulating their query; it also benefits users who simply want to explore what is inside the database.
- Context sensitive - It provides ways for users to specify or choose their context (i.e. search target and constraint) when creating the visual output that they desire.
- Diversified - It diversifies the results and groups the results of same structure, which can satisfy different users' search intentions.

In the rest of this paper, we firstly introduce VSO and then illustrate how users can operate on VSOs to explore the database, and how such exploratory search paradigm helps address each unique challenge of XML search as compared to web search (in Sec. 2). Then we present our system architecture to support different users' information needs in terms of index design and information retrieval methods (in Sec. 3). Lastly, we showcase how ClearMap helps address each of the aforementioned challenges (in Sec. 4).

# 2. ADDRESSING VARIOUS USER INFORMATION NEEDS

In this section, we present typical scenarios that represent different user information needs, and show how a combination of our interactive exploratory search paradigm with our previous work can help address each of them. In ClearMap, we focus on three critical yet commonly encountered scenarios: what users search for is available in the data, users have a general interest (but no specific knowledge) to explore and learn from the database by keyword query, and what users search for is unavailable in the data. These three scenarios are also a snapshot of the three facets of exploratory search: lookup, learning and understanding [9]. Accordingly, we will showcase our solution to each problem in Sec. 4.




(a) VSO 1 (b) VSO 2

**Figure 1: Two Sample Visualized Search Objects (VSOs)**

## 2.1 Visualized Search Object

Before introducing various information needs and challenges, we present a central concept called Visualized Search Object (VSO), around which we organize, retrieve, visualize and manipulate the results of a keyword query, and further enable an interactive exploration of the database by exploiting the inter-connectivity of all query results in a database.

DEFINITION 1. *Given a keyword query $Q$, a matching VSO of $Q$ is in form of a hierarchical tree, which includes*

- *an associated search target node $st$,*
- *a hierarchical subtree with structure context and the content of each node encapsulated,*
- *the index information of the siblings and parent of $st$ in the XML data,*
- *the associated coordinates used for display on screen.*

Note that, the search target is essentially a node type $T$ as later discussed in "*Search Target Identification*" of this section. A search target node $st$ is a node of type $T$ in the XML data, and the query result is a subtree rooted at the search target node $st$.

VSO forms a basic representation of the result, and possess the following properties:

- *Mergeable*: VSOs with the same search target and compatible structure [13] can be merged as one single VSO.
- *Diversified*: for any two VSOs with different structures but matching a certain keyword query, the differences are highlighted for easy distinction.
- *Navigational*: users can do a navigational browsing to another VSO which is a sibling, child or parent of the current VSO visited. As a result, users can navigate any part of the data due to the inter-connectivity of nodes in XML data.

EXAMPLE 1. *Fig. 1 shows two sample VSOs for the keyword query "postman" on IMDB dataset*[1]*. Fig. 1(a) and Fig. 1(b) correspond to two different search targets even though they both contain "postman"; such difference is used to diversify the search results.*

[1]IMDB is a database containing movies' information like title, rating, director, actor, actress, etc. http://www.imdb.com/interfaces

*Note that they do not have the same target or compatible structure, so they cannot be merged in this case. The VSO in Fig. 1(a) corresponds to a subtree rooted at the search target which is a* person *node. Besides the target, a VSO records the subtree as well as the index to its two sibling nodes and parent node. Such information supports the navigational property, where users can navigate from one VSO to another.*

Since all query results are interconnected in a tree structure [13], the structure context serves as a link among query results. More importantly, each VSO not only visualizes a particular result subtree, but also provides a mechanism for users to manipulate and interact with that result, to further explore all related information (in the database) connected with that result, and finally to find their desired information without reformulating the current query. This is achieved by maintaining for each VSO the index information of its siblings and parent, which we will discuss in 3.1.

## 2.2 Scenario 1: What Users Search For is Available in the Data

First, let us look at how traditional XML keyword search handles users' queries. XML data is usually modeled as a rooted tree. Existing works mainly focus on defining the matching semantics, i.e. what should be returned as results, and designing efficient retrieval methods for a particular matching semantic. A basic one is to find the subtree whose root node is the lowest common ancestor (LCA) of all query keywords, upon which various semantics such as Smallest LCA (SLCA) [5], Exclusive LCA (ELCA) [11] and object-oriented LCA [10] are proposed. We will present how our proposed interactive and exploratory search paradigm enhances the search in the following four aspects, while users do not need to reformulate their queries.

### *2.2.1 Search Target Identification*

The first and foremost challenge in database keyword search is the identification of search target, while in web search it is simply the web document. Without figuring out the search target of a user query, the matching results associated with different search targets are messed up together, which badly annoys users in result consumption. Since XML data contains both structural and content information, queries usually contain various ambiguities [2]:

1. A keyword can appear both as an XML tag name and as a text value of some other nodes.
2. A keyword can appear as the text values of different types of XML nodes and carry different meanings.
3. A keyword can appear as an XML tag name in different contexts and carry different meanings.

The keyword ambiguity problem may lead to various interpretations of the search target and constraint. Therefore, we first distinguish the type of a node in XML data by its prefix path from root node, and the search target is referred as node type. Next, we propose guidelines to capture human intuitions in measuring the confidence of a certain node type $T$ as the desired search target of a query $Q$. A desired target node should be related to every query keyword and contain enough relevant yet non-overwhelming information (details can be found in [2]). As a result, we provide a list of promising targets $T$ for user to choose, after which we proceed to retrieve the result as a subtree rooted at $T$.

### *2.2.2 Result Diversification*

The ambiguity may cause not only various search targets, but also various 'search constraints' (implicitly expressed in a keyword query). However, existing works do not distinguish such constraints in result retrieval or ranking stage. Again, much user time has to be spent on consuming results, especially when the desired results are ranked far behind. Therefore, instead of driving users to reformulate their queries, we can achieve better user experience by grouping the query results by different search constraints. Since results are of hierarchical structure, we classify and quantify different search constraints by capturing the confidence of a node $n$ to be searched via (as a constraint) and the structural relationship of nodes, as illustrated in our earlier work [2]. Then the query results can be grouped, and users can see a high-level result group (in form of VSOs) before they choose to zoom in to see all results matching a certain search intention.

### *2.2.3 Result Visualization*

We also find that existing search methods all return a list of independent subtrees as query results. However, all the data in an XML tree are *interconnected* by the hierarchical structure. Each query result thus is a part of the XML tree, and many of them may have sibling or containment relationship in the global context of the whole XML database. Without showing such relationship, the results alone could be misleading or less comprehensible. For example, suppose there is a subtree containing the information of a pencil, such a subtree may not contain all the important information of the pencil, like category, because category node could appear as a parent node of that subtree in the XML database. Simply showing subtrees as independent subtrees could be misleading, say a pencil in *make-up* category is very different from a pencil in *stationary* category. Therefore, how to visualize the query results for XML keyword search properly is a crucial part in enhancing the search experience. In particular, how to support zoom in/out for users to view results at various granularity of details and how to dynamically load the data in user's device window are two challenges.

In order to maximize the usefulness of visualization, we combine the VSO defined in this paper with our map-styled visualization of the XML data [13], in order to visualize the query results within the global context of the whole XML database. In such a manner, users can easily tell the difference and see the relationship among the query results. Furthermore, when visualizing all results of a query in the screen, we merge those visualized search objects (VSO) that are compatible (i.e. having the same search target and structure context), by utilizing the *mergeable* characteristic of VSO.

## 2.3 Scenario 2: Interactive Result Exploration & Data Navigation

Besides complementing the traditional lookup task in scenario 1, another important type of information need from user is to learn from the data or the result for a better understanding of the data they are playing with. Learning is an iterative process that requires a search-refine paradigm, but with exploratory search facility provided, users can get rid of such reformulating process.

In particular, there are at least two cases to consider.

*Case (i)*: Users have a general interest but no specific knowledge on the data, and intend to explore and learn through keyword query. For instance, they may try to learn the schema or structure of the database from the results of their keyword query.

*Case (ii)*: When users finish one round of search, it is also common that users may want to further explore more information of a particular query result. For example, users who finished searching for a particular laptop may also want to further search for the shops selling such a laptop or other items sold by this shop.

Unfortunately, such interactive exploration is more or less ignored by existing XML keyword search techniques, which usually handle one round of "lookup" search. If users need to further explore a particular query result in the previous round of search, they

have to reformulate the original query and re-submit the new query (at least once). In contrast, recall Definition 1 the VSO stores the reference to its parent and siblings, which facilitates the exploration from one result to another (for case (i)); meanwhile, the inter-connectivity of the VSOs and the aforementioned result visualization module enable a natural navigation of the underlying data (for case (ii)). For instance, the information related to a particular result subtree appears right above it, connected by the hierarchical tree structure of the XML data, and a simple upward navigation operator serves the purpose.

## 2.4 Scenario 3: What Users Search For is Unavailable in The Data

In this scenario, a search engine may return an empty result or even worse, return erroneous mismatch results because what users search for is unavailable in the data. E.g., a list of mismatch results are returned when what users search for is unavailable [12]. We proposed to take user interaction to improve the user experience by providing the following three categories of information to guide users based on our earlier work [12].

**(1) Notification.** It is necessary to give a notification "what you search for is unavailable" to users. Otherwise, users have to wade through the mismatch results even to realize that what they search for is unavailable. Existing XML keyword search semantics are all based on LCA, which is trying to find the subtree whose root node is the lowest common ancestor of all query keywords. As long as all the query keywords appear in the XML data, it can always find some subtrees as results according to the LCA semantics. Even what users search for is unavailable in the XML data, some mismatch results will be returned. E.g., if users want to search for a particular laptop model *Vaio W* with red color by issuing a query "Vaio W red", and somehow red color is unavailable for that laptop model in the data. Mismatch results will be returned with "Vaio W" matches one laptop and "red" match another laptop. We proposed a detection approach in [12] by verifying the query results at schema level for detecting the mismatch results.

**(2) Explanation.** An explanation on which keyword(s) lead to empty result or mismatch result can greatly help users understand how the problem occurred. E.g., if a user wants to find a laptop model *Vaio W* with red color by issuing a query "Vaio W red", and somehow red color is unavailable for that model in the data, then it is natural to let the user know which keyword(s) lead the whole query to empty or mismatch result, say the keyword "red". We try to find such explanation by searching for approximate results in the XML data [12]. Then we can discover which keyword(s) is not satisfied in the approximate results and leads to the mismatch results.

**(3) Suggestion.** Another crucial part is suggested queries, which should be generated based on the availability of the data and should not lead to empty result or mismatch results. E.g., if red color is unavailable for a product, some suggested queries by replacing red color to some other available color will be a great help to users. Such suggestion can be found in the approximate results discovered in the previous step.

Following the above framework on how interactive input can help improve XML keyword search, we built ClearMap with source codes public for access at [1]. To the best of our knowledge, it is the first XML keyword search system which can fulfill all the above interaction mechanisms to improve the user experience and overcome the major challenges of keyword search over XML databases.

# 3. SYSTEM ARCHITECTURE

The architecture of ClearMap is shown in Fig. 2. The input of the system is a keyword query issued by users, and the output is a set of results in the form of Visualized Search Objects (VSOs), which provide users the visualized content as well as the structure context of the results (see Definition 1).

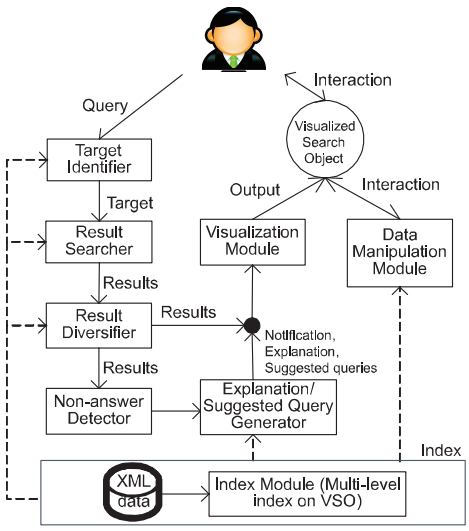



**Figure 2: Architecture of ClearMap**

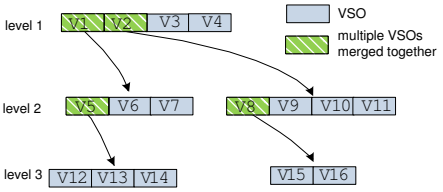



**Figure 3: Multi-level VSO index**

ClearMap comprises several components, each of which is in charge of a particular task in one of the three scenarios aforementioned. The workflow of ClearMap is as follows. First, user's query will be passed to *Target Identifier*, once the possible targets are identified, it interacts with the user to decide the search target that fits to his/her context. Then, it proceeds to *Result Searcher*, which will generate the query results (by any existing XML keyword search method) in the form of VSOs. Next, the query results will be passed to *Result Diversifier* which will cluster the results by different search constraints (at structure level). When results are returned to user, they will be fed to *Non-answer Detector* simultaneously to check whether any mismatch problem occurs. If so, it means there is no desired answer for users' query and it will inform the *Explanation/Suggested Query Generator* to generate notification, explanation as well as suggested queries. After that, both the suggestions and the results of the original query are passed to *Visualization Module*, which is in charge of (VSO) result presentation to users in a precise, interactive and user-friendly way. When users interact with the VSOs to either further explore the query results or navigate the databases, the interaction is captured and sent to *Exploration Module*, which will retrieve the necessary data dynamically from the data and index at server side.

## 3.1 Challenges in Index Building

Since ClearMap integrates many different solutions, each of which works on standalone indexes, it poses the first challenge, i.e. how to build a one-size-fits-all *Index Module* such that relevant information requested by different components of the system architecture can be efficiently retrieved. To tackle the fist challenge, we analyze each component and extract the common index information shared

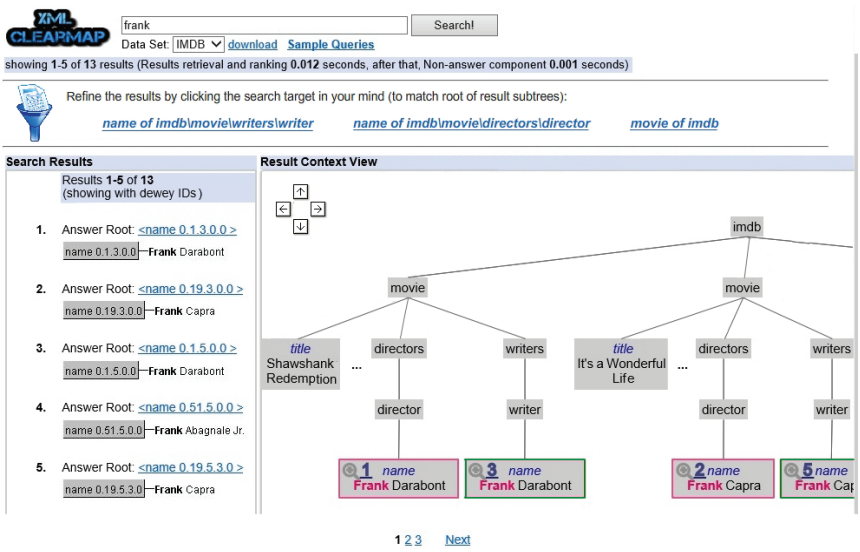



**Figure 4: Interactive Exploratory Search Paradigm for Scenario 1, i.e. what users search for is *available***

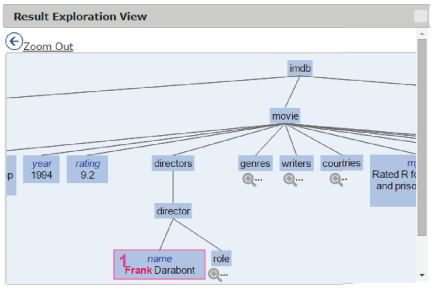



**Figure 5: Interactive Exploratory Search Paradigm for Scenario 2, i.e. interactive result exploration and data navigation**

by more than one component. As a result, the *Index Module* consists of three different indexes: inverted list index, statistics index and visualization index. To further improve the performance, we also build a buffer pool in the *Index Module* to buffer the retrieved information for subsequent reuse. E.g., given a query, *Target Identifier* and *Result Searcher* may both access the inverted list index for the same set of keywords. Therefore, we buffer the inverted lists retrieved by *Target Identifier* for later reuse by *Result Searcher*.

The second challenge is how to visualize a VSO when a VSO is a huge subtree. For instance, for a VSO corresponding to a movie subtree (in IMDB dataset), it may contain hundreds of VSOs corresponding to actor/actress information. Thus it is inappropriate to show such overwhelming information under this subtree to user. To properly visualize the movie VSO, we can merge all the actor/actress VSOs in order to hide those similar information, by exploiting the mergeable property of VSO. Merged VSOs will be expanded in an interactive way according to user's navigation operator (e.g. click-through to see details, drag down, up, left, right, etc.). In order to index the VSOs and the merged VSOs for efficient retrieval, we build a multi-level VSO index which is similar to an R-tree index. Fig. 3 shows a sample of the multi-level VSO index, where merged VSO $V1$ contains VSO $V6$, $V7$ and another merged VSO $V5$. When users navigate to expand a merged VSO, hidden information can be retrieved following the index from level to level. This idea originates from our previous work [13] in presenting the same results at different granularity, but differs with [13] in that all the output are organized, indexed and operated around VSOs instead of nodes of XML data tree.

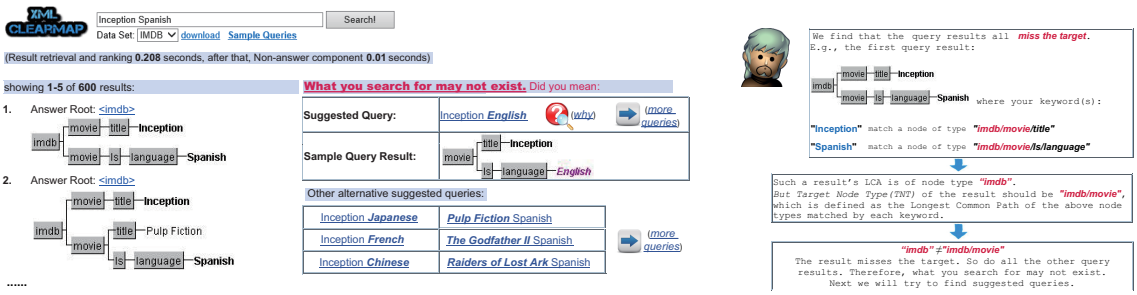
# 4. DEMONSTRATION

It is challenging in itself to define what constitutes an exploratory search, and determine whether an exploratory search system is effective. In this section we will demonstrate how exploratory search with user's interactive input helps address the specific challenges raised in XML keyword search (illustrated in Sec. 2). Since users will undoubtedly be able to tell what works well for them, we very welcome the audience to try our demo and provide their evaluation and feedback (online or offline), for us to further improve the usability of our exploration search paradigm.

In particular, we demonstrate the interaction and visualization of ClearMap in three scenarios, as shown in Figures 4, 5 and 6. Currently two datasets are available, i.e. DBLP and IMDB.

**Scenario 1: what users search for is available in the data** Fig. 4 is a screenshot of the system for a query "frank" issued over IMDB. Traditional methods will find some subtrees containing all query keywords as results. E.g., the left pane of Fig. 4 shows a list of direct results found by the XML search techniques adopted by *Result Searcher*. They are just some *name* nodes containing the keyword "frank". As we can see, simply displaying them is far from enough to meet users' information needs. On the right pane in Fig. 4, the query results are shown in the form of VSO. The relationship among the VSOs are clearly shown to users. This is critical because the query results are actually interconnected in the XML data. Without showing the relationship users could have difficulty in consuming the results. E.g., from the right pane users can know that result 1 and result 2 refer to the name of directors, while result 3 and result 5 refer to the name of writers. Besides, users can also tell that result 1 (director) and result 3 (writer) are actually working for the same movie. These information is critical for users to properly understand the query results.

Right under the search box at the top of Fig. 4, the *Target Identifier* helps identify the promising search targets for user to choose, to further refine the results. E.g. in Fig. 4 we find three promising search targets for query "frank", i.e. writer, director and movie. Since they have different targets, they are highlighted by rectangles with different colors. User can designate the desired one and the visualization will be updated to show only those results under the designated target, otherwise we will keep all promising targets in mind by default. Then *Result Diversifier* will analyze possible search constraints and cluster the results with same constraints in the left pane, highlighted in different colors. All in all, it comes to how to visualize the above targets, constraints, results and even the data itself (for exploration purpose) in an efficient and interactive way. Our visualization has two features: (1) Similar to any Geo-map, we generate several copies of the same XML data with different degrees of detail, allowing users to zoom in/out to check the result/data. Most XML data store many similar-structured data at the same hierarchical level, and keep growing along the horizon-




(a) Non-answer Notification, Explanation and Suggested Queries (b) Explanation after clicking "why" button

**Figure 6: Interactive Exploratory Search Paradigm for Scenario 3, i.e. what users search for is *unavailable***

tal axis. E.g., right below the *imdb* node, the movie items keep growing, leading to a wide XML data tree. In order to provide proper granularity of detail, we propose to merge/expand compatible VSOs to hide/show different level of details, by utilizing the mergeable characteristic of VSO.

**Scenario 2: Interactive Result Exploration & Data Navigation** Users can explore a particular query result by clicking the result ID in the right pane of Fig. 4. An exploration window will be popped up, as shown in Fig. 5. Within the exploration window the whole XML data is available for exploration with that result at the center. The whole exploration process is formed by navigating (by dragging down, up, left and right) and zooming in/out (by clicking the suspension points to zoom in an area for hidden information). E.g., in Fig. 5, users can dragging the display to explore the structure information of result 1, i.e. director Frank Darabont. Users can easily get the structure information, like Frank is a director of a movie, what information is available for the movie, above the movie node is the imdb database root node, etc. Besides, hidden information is represented as suspension points in the display. Users can click the suspension points to zoom in to that subtree, like genres, writers and countries information of that movie. Such an operation will lead the system to load more VSOs dynamically from the server, supported by a R-tree liked index [13].

Besides, a dragging pad is provided for users to move left/right/up/down, to further explore the relationship between query results and XML data. In this way, users with different search intentions can easily adjust the query results to meet their needs without reformulating a new keyword query.

**Scenario 3: what users search for is unavailable in the data** Fig. 6 is a screenshot of the system for a query "Inception Spanish" issued over IMDB dataset. As we can see at the left pane of Fig. 6(a), for all results being returned, "Inception" matches movie title and "Spanish" matches movie language. The search target is very likely to be a movie. However, the movie *Inception* has no Spanish version in our dataset. All results being returned are mismatch results, i.e., the subtrees rooted at *imdb* node, because what users search for is unavailable in the data. To detect mismatch results for XML keyword search is not an easy task, we use the MisMatch solution in [12] for detecting mismatch results, generating notification, explanation and suggested queries. All these helpful information are shown to users, as shown at the right-hand side of Fig. 6(a). Here, one suggested query is "Inception English" that provides an English version of the same movie. Furthermore, users can click the "why" button to seek detailed explanations deriving the suggestions, as shown in Fig. 6(b). Users can also find more suggested queries by clicking the "more queries" button. All the suggested queries are derived from the XML data and guaranteed to have reasonable query results.

The main idea in [12] works as follows. We first infer users' possible search targets for a query $Q$ based on its results $R$; then it will investigate each result $r$ in $R$, to check whether it matches one possible search target. If none of them can match a possible target, we claim that $Q$ has no available result in the data. Hereafter, to generate suggested queries, we propose a TF*IDF-inspired scoring measure to help find "important" keywords in the original query. Then based on each query result $r$, we try to find some "approximate" query results which contain these "important" query keywords and are structurally consistent with $r$, while having reasonable replacement for the "less-important" query keywords. Finally, the suggested queries can be inferred from the approximate results.

**Acknowledgement.** H.V. Jagadish is supported in part by NSF IIS-1250880 and IIS-1017296.

## 5. REFERENCES

[1] *XML ClearMap*. http://xclearmap.comp.nus.edu.sg.
[2] Z. Bao, T. W. Ling, B. Chen, and J. Lu. Effective XML keyword search with relevance oriented ranking. In *ICDE*, pages 517–528, 2009.
[3] G. Koutrika, L. V. S. Lakshmanan, M. Riedewald, and K. Stefanidis. Exploratory search in databases and the web. In *Workshops of the EDBT/ICDT*, pages 158–159, 2014.
[4] F. Li and H. V. Jagadish. Usability, databases, and HCI. *IEEE Data Eng. Bull.*, 35(3):37–45, 2012.
[5] Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD*, pages 329–340, 2007.
[6] G. Marchionini. Exploratory search: From finding to understanding. *Commun. ACM*, 49(4):41–46, Apr. 2006.
[7] A. Nandi and H. V. Jagadish. Guided interaction: Rethinking the query-result paradigm. *PVLDB*, 4(12):1466–1469, 2011.
[8] A. Spink, B. J. Jansen, D. Wolfram, T. Saracevic, and T. Saracevic. From e-sex to e-commerce: Web search changes. *IEEE Computer*, pages 107–109, 2002.
[9] R. W. White and R. A. Roth. Exploratory search: Beyond the query-response paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–98, 2009.
[10] H. Wu and Z. Bao. Object-oriented XML keyword search. In *ER*, pages 402–410, 2011.
[11] Y. Xu and Y. Papakonstantinou. Efficient LCA based keyword search in XML data. In *EDBT*, pages 535–546, 2008.
[12] Y. Zeng, Z. Bao, T. W. Ling, H. V. Jagadish, and G. Li. Breaking out of the mismatch trap. In *ICDE*, pages 940–951, 2014.
[13] Y. Zeng, Z. Bao, T. W. Ling, and L. Li. Malex: a map-like exploration model on xml database. In *KEYS*, pages 32–38, 2012.