Towards Heterogeneous Keyword Search*

Chunbin Lin
University of California
La Jolla, USA
chunbinlin@cs.ucsd.edu

Jianguo Wang
University of California
La Jolla, USA
csjgwang@cs.ucsd.edu

Chuitian Rong
Tianjin Polytechnic University
Tianjin, China
chuitian@tjpu.edu.cn

ABSTRACT

Keyword search is a widely popular mechanism for query processing that alleviates users from understanding complex data structures and learning query languages. Existing keyword search systems are designed and tuned for one specific data model. In big data era, data is usually resident in heterogeneous data sources including unstructured data, semi-structured data and structured data. In order to obtain complete and meaningful results, a system is required to perform keyword search queries upon diverse data sources rather than just one type of data sources. One possible approach is to send the keyword query to all the systems simultaneously, then integrate the returned answers from each data source to obtain final answers. However, answers from different data sources have different formats and may contain duplicates, which yields the *heterogeneous search challenge*.

In this paper we introduce a heterogeneous keyword search system that facilitates integrating answers from diverse data sources. We present several detailed design challenges and also share our preliminary thoughts to the challenges. In particular, we (i) propose a unified result format *entity-relationship pattern* (ERP), (ii) define new ranking functions, (iii) build native index structures, (iv) provide an efficient global top-k processing algorithm, and (v) introduce the fuzzy entity mapping problem. We have built a prototype HKSearch based on our current solutions to support heterogeneous keyword search.

CCS CONCEPTS

• **Information systems** → **Search engine indexing**; *Entity relationship models*; Top-k retrieval in databases;

KEYWORDS

Heterogeneous System, Keyword Search, Entity-Relationship

*This material is based upon work supported by the National Natural Science Foundation of China under grant No.61402329.

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.
ACM TUR-C '17, May 12-14, 2017, Shanghai, China
© 2017 ACM. ISBN 978-1-4503-4873-7/17/05...$15.00
DOI: http://dx.doi.org/10.1145/3063955.3064802

1 INTRODUCTION

Keyword search is a proven user-friendly way of querying data in diverse structures. It allows users to find the information they are interested in without having to learn a complex query language or needing prior knowledge of the structure of the underlying data. The benefits of using keyword search has also been recognized by the database community, who has been applying keyword search into relational databases [1, 4, 7, 11, 12, 14], XML databases [9, 17, 19, 21, 35, 36] and graph databases [10, 13, 14, 18]. Existing keyword search systems are designed and tuned for specific data models. For example, XML keyword search systems, e.g., XRANK [9] and XSEarch [5], successfully answer keyword queries on semi-structured data, while relational keyword search systems, e.g., DBXplorer [1], DISCOVER [11, 12], process keyword queries on structured data. Each system can only deal with one type of data sources. For example, a keyword search system on unstructured data can not answer queries on semi-structured data.

In big data era, data is usually resident in diverse data sources. For example, as shown in Figure 1, the author-paper relationships reside in a relational database, the course information is stored in an XML database, while the paper contents are stored in text documents. Suppose a user wants to explore the information of the professor "Michael Franklin" on the "database" topic. In order to get complete answers, she has to issue the query "michael, franklin, database" to all the systems independently, which is time-consuming. Although she can get some results from some systems, the results are hard for her to understand, integrate and explore as they have different formats and are unorganized. For example, relational keyword search systems return a list of tuples while XML keyword search systems may return a subtree rooted at some LCA nodes. We say she is facing a *heterogeneous search challenge*.

Heterogeneous search challenges are widespread in the enterprise and are typically solved with one of the two following approaches: (i) perform a manual integration for the results gathered from different data sources; and (ii) apply existing data integration algorithms, e.g., [15, 26], to map all the data sources into one structure[1]. The first method is obviously inefficient. The second one changes the underlying data models, which is impossible in many scenarios. For example, different data sources are maintained by different departments, which have different access privileges to each data source. Modifying all the data sources is unrealistic.

[1] For example, we can map all the data sources into relational databases.

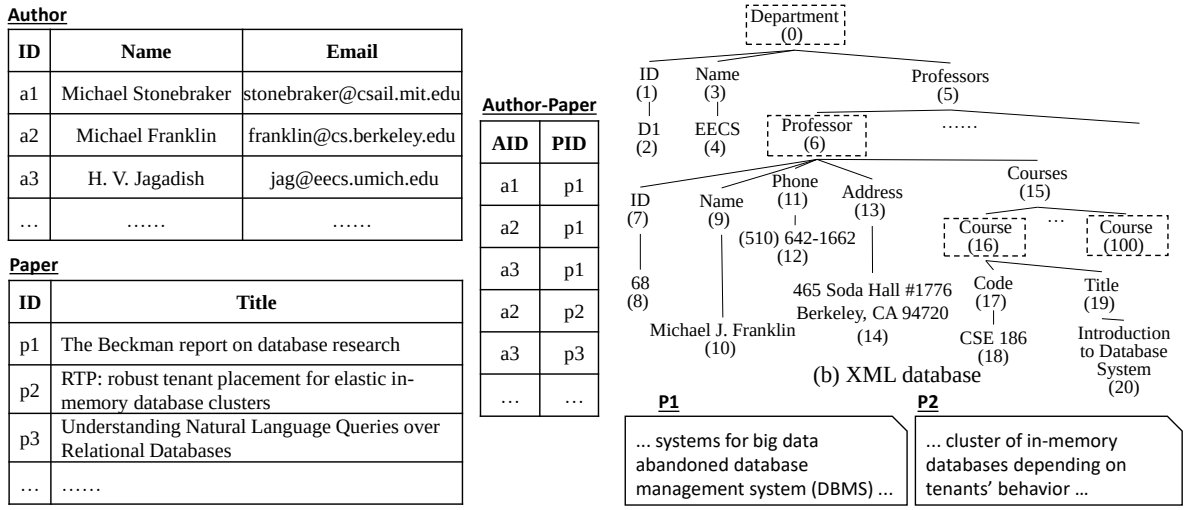

Figure 1: Data in three different sources, i.e., (a) relational database, (b) XML database, and (c) text documents.

In this paper, we try to build a heterogeneous keyword search system, which can successfully integrate answers automatically from diverse data sources without modifying the underlying data models. In order to build such a system, we face the following five challenges, which dissect the heterogeneous search challenges:

(i) *Unified result format.* In order to provide efficient integration for partial answers from different types of data sources, a unified result format should be defined. It should be powerful enough to represent unstructured data, semi-structured data and structured data. In addition, it needs to capture entities and relationships information.

(ii) *New ranking functions.* As a complete answer is integrated by partial answers from diverse data sources, each of which should have its own ranking mechanisms to measure the importance and relevance of the partial answers. To rank the final answers, an additional global ranking function is also necessary.

(iii) *Native index structure.* In order to produce the results in a unified format efficiently, native index structures should be provided, since existing indexes, e.g., traditional inverted index, are infeasible in this scenario.

(iv) *Global top-k processing.* A final answer consists of many partial answers from different data sources. How to avoid constructing all the possible final answers to get top-k final answers is a challenge. This is because the partial answers are not the same, which results in the failure of all the existing top-k algorithms.

(v) *Fuzzy entity mapping.* Partial answers from different data sources may contain duplicate entities and attributes, which should be eliminated. The entities with exact same values are easy to be identified. However, values with different expressions but referring to same entities are hard to be discovered. In order to detect them, we need to both check the syntactic similarities and semantic similarities.

We share our preliminary thoughts and solutions for the above five challenges. Based on the initial ideas, we build a prototype system called HKSearch. The rest of this paper first introduces existing keyword search systems/algorithms (Section 2), then presents the architecture of our system in Section 3. We discuss the five challenges in details in Section 4. Finally, we conclude this paper in Section 5.

2 RELATED WORK

2.1 Keyword search on unstructured data

The topic of keyword search over text documents, e.g., HTML documents, has been studied extensively[6, 16, 28, 37]. They always output a list of individual pages containing all the keywords as results. In the case that there are no pages that contain all the keywords, pages with some of the input keywords are returned and ranked based on the relevancy. However, the existing methods cannot integrate interrelated pages into one relevant and meaningful answer. To solve the problem, EASE [18] models linked web pages as graphs, then returns Steiner graph as results, which organizes relevant pages together.

2.2 Keyword search on semi-structured data

A number of studies have been done on supporting keyword search on XML databases. In particular, XRANK [9] proposes a stack based algorithm to efficiently compute LCAs. XKSearch [35] defines Smallest LCAs (SLCAs) to be the LCAs that do not contain other LCAs. Meaningful LCA

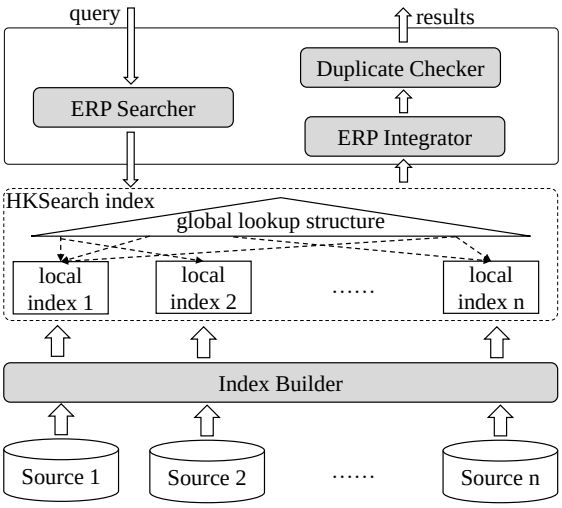

Figure 2: The architecture of HKSearch.

(MLCA) [19] incorporates SLCA into XQuery. VLCA [17] and ELCA [36] introduces the concept of valuable/exclusive LCA to improve the effectiveness of SLCA. MESSIAH [29] handles cases of missing values in optional attributes. These works do not consider the semantics from keyword query. XSeek [23] and XReal [3] consider the semantics of entities, but ignore the relationships. XKeyword [13] exploits semantics via the XML schema. None of the existing XML keyword search methods can process queries over heterogeneous data sources.

2.3 Keyword search on structured data

Keyword search in relational databases [1] has recently emerged as a new research topic. The representative relational keyword search systems are DBXplorer [1], DISCOVER [11, 12], BANKS-I [4], BANKS-II [14] and EASE [18]. They return tuple trees/graphs as answers for a given keyword query. In particular, DISCOVER and DBXplorer generate trees of tuples connected through primary-foreign key relationships that contain all of the input keywords. BANKS identifies connected trees in a labeled graph by using an approximation of the Steiner tree problem. DISCOVER-II considers the problem of keyword proximity search in terms of disjunctive semantics, as opposed to DISCOVER-I which only considers conjunctive semantics. BANKS-II uses a bidirectional strategy to improve the efficiency of keyword search over graph data. However, it still needs to identify Steiner trees from the whole graph. To improve the performance, [7] employs a dynamic programming method, [10] proposes a partition-based method, while EASE [18] proposes r-radius Steiner graphs, which only searches meaningful Steiner graphs with acceptable sizes.

2.4 Heterogeneous keyword search

None of the above systems can solve the heterogeneous search challenges, as they are all designed and optimized for specific data models. The only relevant existing work is EASE [18]. The goal of EASE is to propose an adaptive search algorithm running on heterogeneous data sources. However, EASE does not solve the heterogeneous search challenges essentially as it ignores the scenarios that answers from different data sources need to be integrated. In particular, EASE is unable to integrate search answers from each data source, while HKSearch can. This yields some limitations of EASE, e.g., it lacks of global ranking functions and does not considering the duplication elimination problem.

Though HKSearch is different from EASE, HKSearch combines the index structure (i.e., EI-index) of EASE due to its high performance. More precisely, HKSearch combines the EI-Index and a native index on semi-structured databases together to produce a native index structure, which will be discussed fully in Section 4.3.

3 ARCHITECTURE

Figure 2 gives an overview of HKSearch architecture. It has two parts: *HKSearch Index Builder* and *HKSearch Query Processor*.

HKSearch Index Builder. The *HKSearch Index Builder* builds one HKSearch index for all the data sources. The HKSearch index has two components: one global lookup structure and several local indices (one local index for one data source). The global lookup structure maintains pointers referring to corresponding local indices for each search keyword[2]. Local indexes have different structures. For unstructured and structured data sources, the local indexes are similar to the EI-index in EASE [18]. For semi-structured data sources, we introduce a new index by using keywords as keys and the labels of entity nodes as values. The details of HKSearch index is introduced in Section 4.3.

HKSearch Query Processor. Once a new query is received, the *ERP searcher* retrieves the index structure to produce a list of ranked ERPs from each local index (ERP is defined in Section 4.1). Then the *ERP Integrator* applies a top-k algorithm (the top-k algorithm is proposed in Section 4.4) to get k final ERPs with highest overall scores (the ranking function is defined in Section 4.2) by carefully accessing ERPs in each list. Finally, the *Duplicate Checker* eliminates duplicate entities by applying fuzzy mapping method, which is studied in Section 4.5.

4 CHALLENGES

In this section, we describe the challenges mentioned in Section 1 and give our preliminary thoughts.

4.1 Unified Result Format

A keyword answer in the heterogeneous environment is integrated by multiple partial answers from different data sources. In order to integrate them without any burden, a unified result format is necessary. It is challenging to define a unified

[2]In the global lookup structure, we index all the distinct keywords in the whole domain crossing all the data sources. In other words, the domain is the union set of all the distinct words.

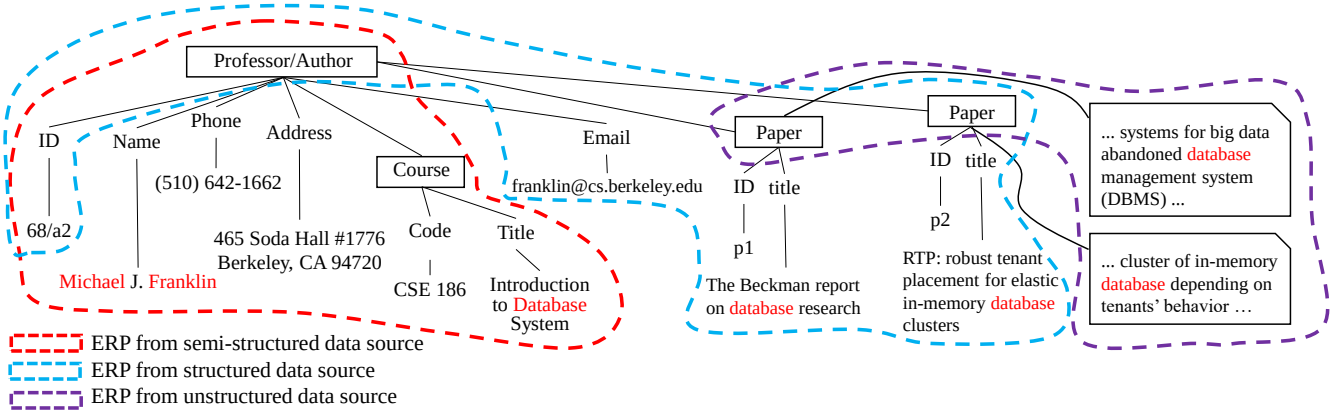

Figure 3: An ERP of query "michael, franklin, database" returned by HKSearch, which is constructed by three ERPs from unstructured data (highlighted in purple), semi-structured data (highlighted in red), and structured data (highlighted in blue).

result format as this format should be powerful enough to express answers in unstructured data, semi-structured data and structured data. In addition, this format should capture entitles and the relationships among entities. Therefore, we define a unified result format *entity-relationship pattern (ERP)*, which is a graph model.

Entity-relationship pattern (ERP). Given a query $q = q_1, ..., q_k$, an ERP is a Steiner Tree [12, 18] covers all the matched-entities and all the nodes in the ERP are entities. The entities in unstructured data (e.g., text documents), semi-structured data (e.g., XML databases) and structured data (e.g., relational databases) are documents, elements and tuples respectively. The edges in an ERP are hyperlinks, parent-child relationships and primary-foreign-key relationships respectively. In addition, the matched-entity is defined as follows:

Definition 4.1 (Matched-entity). Given a keyword term k, an entity E is a matched-entity if the values of its attributes contain k.

For example, consider a keyword term "Michael", the existing XML keyword search systems return the node "Michael J. Franklin(10)" in Figure 1(b) as a result. However, it is not an entity (so not a matched-entity either). The node "Professor(6)" is a matched-entity in HKSearch, as it is an entity node and "Michael J. Franklin(10)" is the value of its attribute "Name(9)". The answer to the query "michael, franklin, database" in HKSearch is shown in Figure 3, which integrates ERPs from diverse data sources.

4.2 Ranking Function

As the final answers are integrated from different data sources. Each data source has its own features. For example, XML databases have structure information. So for each data source, we need its own ranking function. And for the final answers, we need a global ranking function.

Local ranking function.

Here, we define local ranking functions for each kind of data source. In particular, we use IR ranking for unstructured data, XML ranking for semi-structured data and DB ranking for structured data.

IR Ranking. Given a query $q = \{q_1, ..., q_k\}$ and an ERP $\mathcal{E}$ produced from unstructured data sources. The score $\mathcal{F}_{IR}(\mathcal{E}, q)$ of $\mathcal{E}$ to query q is defined as follows:

$$\mathcal{F}_{IR}(\mathcal{E}, q) = \sum_{i=i}^{k} \mathcal{F}_{IR}(\mathcal{E}, q_i) = \sum_{i=i}^{k} \frac{tf(\mathcal{E}, q_i) \times idf(q_i)}{|\mathcal{E}|}$$

where $tf(\mathcal{E}, q_i) = 1 + ln(1 + ln(1 + f(\mathcal{E}, q_i)))$ and $idf(q_i) = \frac{N+1}{N_{q_i}+1}$. $f(\mathcal{E}, q_i)$ is the term frequency of term q_i in ERP $\mathcal{E}$, N is the number of all the ERPs and N_{q_i} is the number of ERPs containing term q_i.

XML Ranking. Given an ERP $\mathcal{E}$ for a query q in semi-structured data, we define the function $\mathcal{F}_{XML}$ to measure the score of $\mathcal{E}$ for q:

$$\mathcal{F}_{XML}(\mathcal{E}, q) = \sum_{i=i}^{k} \mathcal{F}_{XML}(\mathcal{E}, q_i) = \sum_{i=i}^{k} \frac{xmltf(\mathcal{E}, q_i) \times xmlidf(q_i)}{|\mathcal{E}|}$$

where $xmltf$ and $xmlidf$ are functions of XML TF and XML IDF, as defined in [3].

DB Ranking. To calculate the score of an ERP $\mathcal{E}$ for a query q in structured data, we use the function $\mathcal{F}_{DB}$:

$$\mathcal{F}_{DB}(\mathcal{E}, q) = \sum_{1 \le i < j \le k} S((q_i, q_j), \mathcal{E}) \times (\mathcal{F}(\mathcal{E}, q_i) + \mathcal{F}(\mathcal{E}, q_j))$$

where $\mathcal{F}(\mathcal{E}, q_i)$ and $\mathcal{F}(\mathcal{E}, q_j)$ have been defined above. Function $S((q_i, q_j), \mathcal{E})$ measures the distance of q_i and q_j in $\mathcal{E}$. We adopt the Equation 7 defined in [18] to compute $S((q_i, q_j), \mathcal{E})$.

Global ranking function. In the above, we define the local ranking functions for different data sources. Given a set of ERPs $\mathcal{E}_1, ..., \mathcal{E}_n$ returned from diverse data sources, a final ERP is constructed by mapping them together. Therefore, we need to define a global ranking function $\mathcal{G}(\mathcal{E}, q)$ to compute the scores for final ERPs.

$$\mathcal{G}(\mathcal{E}, q) = \sum_{i=1}^{n} \alpha_i \times \mathcal{F}_i(\mathcal{E}, q)$$

where $\sum_{i=1}^{n} \alpha_i = 1$ ($\alpha_i \geq 0$ for $i \in [1, n]$) and $\mathcal{F}_i \in \{\mathcal{F}_{IR}, \mathcal{F}_{XML}, \mathcal{F}_{DB}\}$. The challenge is to assign right α to each local function. To simplify the ranking function, in HKSearch, we set $\alpha_1 = \alpha_2 = ..., \alpha_n = 1/n$.

4.3 Index Structure

To support heterogeneous keyword search efficiently, we propose a native index structure, which has two main components: a global lookup structure and several local indexes pointed by values in the global lookup structure.

Global lookup structure. The HKSearch global lookup structure is a hash structure with keywords as keys, and the value of each key k is a pointer-array $\mathcal{P}_k$ with size N, where N is the number of total data sources. $\mathcal{P}_k[i]$ stores the pointer pointing to the local index $\mathcal{I}_i$.

Local index structure. Each data should source return a list $\mathcal{L} = \{(\mathcal{E}_1, s_1), ..., (\mathcal{E}_n, s_n)\}$, where $\mathcal{E}_i$ is an ERP and s_i is its corresponding score, which is computed by using the ranking functions defined in Section 4.2. As mentioned before, for unstructured data and structured data, the ERP is the same to the Steiner tree in [12, 18]. Therefore, for the unstructured data and structured data, we modify the EI-index in [18] by applying our ranking functions to calculate scores for ERPs as local indexes.

However, for semi-structured data, the EI-index is infeasible, as it is still an LCA-based method, which may not return entities. To solve the problem, we do the following changes: (i) Identify all the entities in an XML tree, which can be done by either using the given schema (if provided) or checking its siblings. For example, [23] regards a node as an entity node if it has siblings of the same name. All the entities in Figure 1(b) are highlighted in dotted rectangles; (ii) Assign JDewey labels [13] only to entities; (iii) For each term k, build an inverted list which contains the labels such that the corresponding entity nodes either contains k or it is the lowest ancestor of the nodes containing k.

4.4 Top-k Query Processing

Users are more interested in top-k answers rather than a complete list of answers to the given query. As a complete ERP $\mathcal{E}$ is constructed from $\{\mathcal{E}_1, ..., \mathcal{E}_n\}$, where $\mathcal{E}_i$ is an ERP returned from the i-th data source. One straightforward method to get the k final ERPs with highest overall scores is to (i) construct all the possible ERPs, (ii) compute the scores for all the ERPs, (iii) rank the ERPs based on the scores in a decreasing order, and (iv) return the top-k ERPs. However, this method accesses more ERPs than needed. In addition, existing top-k algorithms, e.g., TA [27] and NRA [8], cannot be applied as they all assume that the elements in lists should share same identifiers. However, in our case, the ERPs in different data sources are different. Therefore, we need to propose new top-k query processing algorithm to efficiently get final top-k ERPs.

To solve the challenge, we propose a top-k algorithm, which greedily accesses the ERPs in lists. More precisely. Given a set of lists $\mathcal{L}_1, ..., \mathcal{L}_n$, where $\mathcal{L}_i = \{(\mathcal{E}_1, s_1), ..., (\mathcal{E}_n, s_n)\}$ is a list of (ERP, score) pairs. HKSearch operates on the lists to get top-k answers in the following steps:

(i) Maintain a pointer p_i for each list $\mathcal{L}_i$. Initially, p_i points to the first $(\mathcal{E}_i, s_i)$ pair;

(ii) Construct a final ERP $\mathcal{E}$ using the current accessing ERPs $\mathcal{E}_1, ..., \mathcal{E}_n$ and computes the score s for it by using the global ranking function over $s_1, ..., s_n$. Then put $(\mathcal{E}, s)$ to a result set $\mathcal{R}$;

(iii) Move the pointer p_i one step down, where p_i is the pointer of the list $\mathcal{L}_i$ such that the current score s_i is the maximal;

(iv) Continue the second and the third steps until $|\mathcal{R}| = k$, then output $\mathcal{R}$ as final results.

4.5 Fuzzy Mapping

When integrating ERPs produced from different data sources, combining (or eliminating) duplicate attributes and entities is a challenge. In particular, we need to find the values with high similarity (syntactically and/or semantically), as values referring to same entities in different data sources may have different expressions. For example, "Michael Stonebraker" is identical to "Michael R. Stonebraker" since "R." is the middle name and is usually omitted in some systems. Another example is that "Very Large Databases" is the same to "VLDB" since the former one is the full expression of the later one. The first example can be discovered by checking the syntactic similarities, e.g., the number of common words or q-grams [31]. The second example is harder than the first one, as it requires to check the semantic similarities, e.g., synonyms [24].

In the following, we focus on discussing the syntactic similarities and semantic similarities.

Syntactic similarities. To analyze the syntactic similarity for two given strings, two kinds of similarity functions can be utilized: (1) token-based similarity, e.g., Dice similarity, Cosine similarity and Jaccard similarity; and (2) character-based similarity, e.g., edit distance. [30–32] give signature-based methods to find similar string pairs, which can be utilized in this scenario.

Semantic similarities. For cases where syntactically different strings can represent the same real-world entity. For example, "Bill" is a short form of "William". To explore the semantic similarities, existing works [2, 24, 25] partially solve the challenge by applying synonym rules to evaluate the maximal similarities. As this problem is proven to be NP-hard, there is no efficient accurate algorithms. However, the existing works do not provide greedy algorithms with content bound. This is still an open problem.

In HKSearch, we employ the existing methods to find duplicates by setting the threshold to be 0.9. Note that, it may remove some false negative duplicates. In the future, we will improve the accuracy and efficiency of identifying attributes with high semantic similarities.

4.6 Additional Challenges

In this part, we discuss additional challenges and our initial thoughts. We focus on answering the following two questions: (Q_1) how to support heterogeneous keyword search on limited memory? and (Q_2) how to incorporate with modern processors, e.g., SIMD, to speed up the performance?

Answer to Q_1. To support heterogeneous keyword search on limited memory, we apply data compression schemes to indexes. In this paper, we consider the compression methods that not only reduce the space cost, but also not sacrifice the performance. Therefore, we employ the bitmap compression schemes [20, 22, 34], e.g., EWAH and CONCISE. These bitmap compression schemes support Bitwise operators, e.g., AND and OR, directly on compressed data. In the future, we will try other compression methods like PforDelta [38] and MILC [33], which are widely utilized in information retrieval area.

Answer to Q_2. To speed up the performance, we need to carefully design the index structure in order to be compliant to CPU cache lines and SIMD instructions. An SIMD instruction operates on s-bit data where s depends on different processors. Typically, s is 128. The benefit of using SIMD instructions is to improve performance by processing multiple elements at a time. The goal of the cache-conscious design is to reduce cache misses by ensuring that a cache line brought from the memory is fully utilized before it is being evicted.

5 CONCLUSION

We have built a prototype system that supports heterogeneous keyword search over diverse data sources by using our initial preliminary methods. In the future, we will continue to solve the proposed challenges to enhance the system to achieve higher performance and better answer quality.

REFERENCES

[1] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. 2002. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE*. 5–16.

[2] Arvind Arasu, Surajit Chaudhuri, and Raghav Kaushik. 2008. Transformation-based framework for record matching. In *ICDE*. 40–49.

[3] Zhifeng Bao, Tok Wang Ling, Bo Chen, and Jiaheng Lu. 2009. Effective xml keyword search with relevance oriented ranking. In *ICDE*. 517–528.

[4] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and Shashank Sudarshan. 2002. Keyword searching and browsing in databases using BANKS. In *ICDE*. 431–440.

[5] Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. 2003. XSEarch: A semantic search engine for XML. In *VLDB*. 45–56.

[6] W Bruce Croft, Donald Metzler, and Trevor Strohmann. 2010. *Search engines*. Pearson Education.

[7] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. 2007. Finding top-k min-cost connected trees in databases. In *ICDE*. 836–845.

[8] Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *JCSS* 66, 4 (2003), 614–656.

[9] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. 2003. XRANK: Ranked keyword search over XML documents. In *SIGMOD*. 16–27.

[10] Hao He, Haixun Wang, Jun Yang, and Philip S Yu. 2007. BLINKS: ranked keyword searches on graphs. In *SIGMOD*. 305–316.

[11] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. 2003. Efficient IR-style keyword search over relational databases. In *VLDB*. 850–861.

[12] Vagelis Hristidis and Yannis Papakonstantinou. 2002. Discover: Keyword search in relational databases. In *VLDB*. 670–681.

[13] Vagelis Hristidis, Yannis Papakonstantinou, and Andrey Balmin. 2003. Keyword proximity search on XML graphs. In *ICDE*. 367–378.

[14] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. 2005. Bidirectional expansion for keyword search on graph databases. In *VLDB*. 505–516.

[15] Gerti Kappel, Elisabeth Kapsammer, and Werner Retschitzegger. 2004. Integrating XML and relational database systems. *WWW* 7, 4 (2004), 343–384.

[16] Yuangui Lei, Victoria Uren, and Enrico Motta. 2006. Semsearch: A search engine for the semantic web. In *EKAW*. 238–245.

[17] Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. 2007. Effective keyword search for valuable lcas over xml documents. In *CIKM*. 31–40.

[18] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. 2008. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*. 903–914.

[19] Yunyao Li, Cong Yu, and HV Jagadish. 2004. Schema-free xquery. In *VLDB*. 72–83.

[20] Chunbin Lin, Benjamin Mandel, Yannis Papakonstantinou, and Matthias Springer. 2016. Fast in-memory SQL analytics on typed graphs. *Proceedings of the VLDB Endowment* 10, 3 (2016), 265–276.

[21] Chunbin Lin and Jianguo Wang. 2017. SpiderX: Fast XML Exploration System. In *WWW*.

[22] Chunbin Lin, Jianguo Wang, and Yannis Papakonstantinou. 2016. Data Compression for Analytics over Large-scale In-memory Column Databases (Summary Paper). *CoRR* abs/1606.09315 (2016).

[23] Ziyang Liu and Yi Chen. 2007. Identifying meaningful return information for XML keyword search. In *SIGMOD*. 329–340.

[24] Jiaheng Lu, Chunbin Lin, Wei Wang, Chen Li, and Haiyong Wang. 2013. String similarity measures and joins with synonyms. In *SIGMOD*. 373–384.

[25] Jiaheng Lu, Chunbin Lin, Wei Wang, Chen Li, and Xiaokui Xiao. 2015. Boosting the Quality of Approximate String Matching by Synonyms. *TODS* 40, 3 (2015), 15.

[26] Imran R Mansuri and Sunita Sarawagi. 2006. Integrating unstructured data into relational databases. In *ICDE*. 29–29.

[27] Surya Nepal and M. V. Ramakrishna. 1999. Query Processing Issues in Image (Multimedia) Databases. In *ICDE*. 22–29.

[28] Gerard Salton and Michael J McGill. 1986. Introduction to modern information retrieval. (1986).

[29] Ba Quan Truong, Sourav S Bhowmick, Curtis Dyreson, and Aixin Sun. 2013. MESSIAH: missing element-conscious SLCA nodes search in XML data. In *SIGMOD*. 37–48.

[30] Jiannan Wang, Jianhua Feng, and Guoliang Li. 2010. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *VLDB* 3, 1-2 (2010), 1219–1230.

[31] Jiannan Wang, Guoliang Li, and Jianhua Fe. 2011. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *ICDE*. 458–469.

[32] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2012. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD*. 85–96.

[33] Jianguo Wang, Chunbin Lin, Ruining He, Moojin Chae, Yannis Papakonstantinou, and Steven Swanson. 2017. MILC: Inverted List Compression in Memory. *PVLDB* 10, 8 (2017).

[34] Jianguo Wang, Chunbin Lin, Yannis Papakonstantinou, and Steven Swanson. 2017. An Experimental Study of Bitmap Compression vs. Inverted List Compression. In *SIGMOD*.

[35] Yu Xu and Yannis Papakonstantinou. 2005. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*. 527–538.

[36] Rui Zhou, Chengfei Liu, and Jianxin Li. 2010. Fast ELCA computation for keyword queries on XML data. In *EDBT*. 549–560.

[37] Justin Zobel and Alistair Moffat. 2006. Inverted files for text search engines. *CSUR* 38, 2 (2006), 6.

[38] Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. 2006. Super-scalar RAM-CPU cache compression. In *ICDE*. 59–59.