Keyword-based access to relational data: To reproduce, or to not reproduce?

Alex Badan, Luca Benvegnù, Matteo Biasetton, Giovanni Bonato, Alessandro Brighente, Stefano Marchesin, Alberto Minetto, Leonardo Pellegrina, Alberto Purpura, Riccardo Simionato, Matteo Tessarotto, Andrea Tonon, and Nicola Ferro

University of Padua, Italy
{alex.badan, luca.benvegnu.2, matteo.biasetton, giovanni.bonato, alessandro.brighente, stefano.marchesin, alberto.minetto, leonardo.pellegrina, alberto.purpura, riccardo.simionato.1, matteo.tessarotto.1, andrea.tonon.3}@studenti.unipd.it
ferro@dei.unipd.it

Abstract. We investigate the problem of the reproducibility of keyword-based access systems to relational data. These systems address a challenging and important issue, i.e. letting users to access in natural language databases whose schema and instance are possibly unknown. However, neither there are shared implementations of state-of-the-art algorithms nor experimental results are easily replicable. We explore the difficulties in reproducing such systems and experimental results by implementing from scratch several state-of-the-art algorithms and testing them on shared datasets.

1 Introduction

Structured data are an intrinsic and pervasive constituent of Big Data. Raw figures estimate the existence of about 14 billion raw tables, coming from 5.4 million schemas and comprising more than 5.4 million attributes [8] and Gartner puts the relational database market at $26 billion with about 9% annual growth for an expected $40 billion market in 2018. The Deep Web alone is estimated to be 500 times the size of the surface or indexable Web [4, 19] while [7] estimates in excess of one billion structured data sets as of February 2011: more than 300 million sources just come from a subset of all English-language HTML tables [8] and HTML lists [12], a total that does not account for the non-English Web.

Structured queries are not end-user oriented and far away from a natural expression of user information needs by means of keywords [6, 20], given that their formulation is based on a quite complex syntax and requires some knowledge about the structure of the data to be queried. Furthermore, structured queries are issued assuming that a correct specification of the user information need exists and that answers are perfect – i.e. they follow the "exact match" search paradigm. On the other hand, end-users are more oriented towards a "best match" search paradigm where their often vague information needs are

progressively refined by the search process itself. Keyword-based access to relational data [21] is the key technology to lower the barriers of access to such huge amounts of structured data since it matches user keywords or their interpretations to the data structures and domains of selected attributes in order to retrieve and rank database tuples according to their degree of relevance to user information needs. Over the past years, these facts triggered the research community and big data technology vendors to put a lot of effort into developing new approaches for keyword search over structured databases and it is still a primary research and industrial concern [1].

One of the major obstacles to the take up of keyword-based access systems is the lack of a fully-fledged reference architecture [3] which allows for a coherent and unifying vision of the different modules comprising such systems. As a consequence, there is a also lack of commonly shared open source platforms implementing state-of-the-art algorithms for keyword-based access to relational data. This situation is very different from the case of *Information Retrieval (IR)* where there are shared platforms openly available such as Lucene[1] or Terrier[2]. On the contrary, for keyword-based access system, you mostly need to re-implement each system from scratch, somehow "wasting" resources in re-developing existing solutions. Moreover, the lack of shared implementations also raises serious concerns about the *reproducibility* [13–15] of the obtained experimental results, since there is no way to know if a new implementation corresponds to the reference one or in which respect a new implementation is better than a reference one.

Therefore, as part of a student project at the course on databases in the master degree in computer engineering at the University of Padua [2], academic year 2015-2016, we considered several state-of-the-art algorithms, we implemented them from scratch, and we experimented them on the shared datasets provided by [10]. The goal is two-fold: (i) to understand the difficulties and pitfalls in implementing these state-of-the-art algorithms; (ii) to compare against the results of [10] for the same algorithms on the same datasets in order to get an idea of how difficult are these results to be reproduced. All the implemented algorithms and the experimental data are available online[3] as open source.

The paper is organized as follows: Section 2 describes the considered algorithms, their implementation and the main issues encountered; Section 3 introduces the experimental setup; Section 4 reports and discusses the experimental results; and, Section 5 draws conclusions and discusses future work.

2 Implemented Keyword-based Search Algorithms

There are two main approaches to keyword search over relational data: *graph-based* approaches model relational databases as graphs where nodes are tuples

[1] `http://lucene.apache.org/`
[2] `http://www.terrier.org/`
[3] `https://bitbucket.org/ks-bd-2015-2016/ks-unipd/`

and edges are foreign-primary key relationships between those tuples; *schema-based* approaches exploit schema information to formulate SQL queries generate from the user keyword queries. For a broader perspective on keyword search over relational databases, please refer to [18, 21].

As discussed in [2], we have implemented several state-of-the-art algorithms, both graph-based and schema-based. Each group of students was responsible for the implementation of a specific algorithm and all the groups worked independently. All the algorithms are implemented in Java using PostgreSQL[4] as relational engine.

BANKS-I *Browsing ANd Keyword Searching I (BANKS-I)* [5] is a graph-based algorithm: it starts from the nodes containing keywords and traverses all the edges in reverse direction, the so-called backward expansion, by using the Dijkstra's single source shortest path algorithm. When multiple paths intersect at a common node r in the graph, the resulting tree with root r is examined to check whether its leaves contain all the user keywords and it is weighted accordingly. Then, it returns the most relevant trees, until a predefined number of results has been reached.

We start with a pre-processing step which creates the required graph in main memory and, then, we use this graph for answering all the different queries. We create this graph in a schema-agnostic way by deducing the structure of a database from its information schema; in this way our implementation is easily portable to different databases. For each node in the graph, we save in main memory only the identifier of a tuple while an external file is used to store the information contained in the tuple itself, in order to avoid further queries to the database. The graph is represented as an hash map, where keys are identifiers node/tuple and values are lists of adjacent nodes, by using the efficient implementation provided by the fastutil library[5].

[5] lacked details on how to efficiently run multiple instances of the Dijkstra's algorithm, even if these aspects considerably affect the memory usage and the execution time. Moreover, some corner cases were not described: for example, trees with nodes containing all the user keywords, where there is no need to run the Dijkstra's algorithm, or trees composed of only a root and a child node, which we decided to consider as valid if both nodes contain a keyword.

We optimize the execution of multiple instances of Dijkstra's algorithm (one for each keyword node) by computing the next node to be returned by the algorithm only when it is actually required. i.e. only when we have to process a result tree to be returned, which typically involves only an handful of nodes instead of executing the Dijkstra's algorithm on the whole graph.

We also developed an improved version of BANKS-I by observing that it tends to return trees containing more nodes than necessary, which are also potentially irrelevant. Therefore, we penalize longer paths by incrementing the weight of each edge proportionally to the depth of the path.

[4] `https://www.postgresql.org/`
[5] `http://fastutil.di.unimi.it/`

BANKS-II *Browsing ANd Keyword Searching II (BANKS-II)* [17] improves over BANKS-I by allowing forward search from potential roots towards other keyword nodes. The algorithm uses two concurrent iterators, called outgoing and ingoing: both iterators use the Dijkstra's single source shortest path approach but the outgoing iterator expands towards other matching sets using the forward expansion while the ingoing iterator implements the same backward expansion strategy of BANKS-I. In addition, BANKS-II prefers expansion of paths with less branching by using a spreading activation mechanism, which assigns an activation score to every explored vertex and prioritizes the most promising paths to avoid wasteful exploration of the graph.

As in the previous case, a pre-processing step creates the required graph in main memory using a schema-agnostic approach. In this case, instead of using adjacency lists, we implement the graph in an object-oriented way by developing our own `Graph`, `Vertex`, and `Edge` classes. In particular, `Vertex` objects are stored in the `Graph` object using a two-layered Java `HashMap`: the name of a table in the database is used as key to a second `HashMap` which associates an unique identifier to a tuple `Vertex`. Since [17] does not mandate a specific way of weighting edges, we assigned uniform weights.

[17] lacked some details on the update of the distances of the reached ancestors and the spread of activations when adding a new node (ATTACH and ACTIVATE methods); we opted for a recursive solution. In our implementation, the outgoing and ingoing iterators explore the `Graph`, by using the Java `TreeSet` class and defining a custom `compare` method to order nodes by their activation. We decided to use `TreeSet`s instead of a priority queue as suggested by [17] to update more efficiently the activation parameter of the nodes already stored in the `TreeSet`. As soon as a new result for the user query is found, i.e. a rooted tree connecting all the matching keywords, we store it in a Java `PriorityQueue` which ranks the results by the overall sum of the weights of the edges in the tree. Finally, to increase performance, we used two heuristic solutions: we set a maximum distance of the results trees (dmax = 8) as suggested by [17] and to avoid wasteful spreading towards the graph we set a minimum threshold value for the activation of a node even if not explicitly stated in [17].

DPBF *Dynamic Programming Best-First (DPBF)* [11] is a graph-based algorithm which relies on minimum cost Steiner trees to merge trees with the same root and different sets of keywords until a tree contains all searched keywords is found. In the pre-processing stage we create the graph from the database, using specific knowledge of the schema, and serialize it to a file for the subsequent processing. The graph is represented using a Java `HashMap` associating each `Vertex` to the list of its adjacent neighbours. Furthermore, we also keep an additional Java `HashMap` where keys are query keywords and values are sets of vertexes associated to those keywords.

As in the case of the other algorithms, also [11] does not specify the actual data structure to be used to represent the graph, even it actually impacts on performances. Moreover, [11] presents just a specific case of graph exploration

and tree merging but it lacks details on the general case, i.e. the most important for reproducibility purposes.

We also developed an improved version of DPBF by modifying the order of the different steps of the algorithm. Indeed, when DPBF creates a new root adding a level to the tree, it then merges the trees containing different keywords under this new root. However, this step produces a large number of trees, some of which containing the very same keywords and representing the same candidate solution, but, in the end, the algorithm will keep only the trees with the lower weight. Therefore, we swap these two operations: first, we merge trees and remove all the duplicate candidate solutions with higher weight and then we add a new root to the trees, as before. In this way we reduce the number of possible found solutions but we converge to the optimal ones faster, especially in the case of complex schemas.

DISCOVER-II DISCOVER-II [16] is a schema-based algorithm which uses IR-style document-relevance ranking strategies. For each table, the IR engine module extracts and ranks the tuple sets corresponding to the query keywords which, in turn, are used to generate the *Candidate Networks (CNs)*, i.e. join of tuples which are potential answers to the issued query. The CN Generator involves a combination of not-free and free tuple sets – the former contains at least one keyword, the latter contains no keywords but allows us to connect not-free tuple sets via foreign-key joins. The final step is to identify the top-k results by combining the top-k results from each CN in a sort-merge manner.

In our implementation, we build the graph using a schema-agnostic approach, as in the previous cases, and use a `Graph` class, which is a container for tuple sets and their primary key/foreign key links. The full text extension of PostgreSQL and generalized inverted indexes (GIN) are used to implement the IR engine module. Moreover, to improve performance, we store the tuple sets for a given query as a materialized view to have them readily available during the subsequent execution of the algorithm. The Execution Engine module receives as input the set of CNs along with the non-free tuple sets, it repeatedly contacts the database to identify the top-k query results. We compute a bound on the maximum possible score of a tuple tree derived from a CN and discard the CN when this bound does not exceed the score of the already produced CNs. The CNs for a user query are evaluated in ascending size order, so that smallest CN, i.e. the least expensive to process and the most likely to produce high-score tuple trees, are evaluated first.

3 Experimental Setup

We adopted the same evaluation framework of [10], which is based on the Cranfield approach [9] widely used in IR. In particular, we used the Mondial and IMDb datasets. Each dataset, available online[6], contains 50 topics assessed using binary relevance.

[6] `https://www.cs.virginia.edu/~jmc7tp/resources.php`

Mondial has a complex schema and consists of 28 relations, containing 17,000 tuples, 56,000 foreign keys and 12,000 unique terms for a total size of 16 Mbyte; IMDb has a simple schema and consists of 6 relations, containing 1,673,000 tuples, 6,075,000 foreign keys, and 1,748,000 unique terms for a total size of 459 Mbyte.

We kept only the topics where all the studied algorithms retrieved at least one result (relevant or not). Therefore, we removed topics 31–36, 38 and 41 from Mondial, that is we used 42 topics, and we removed topics 21–22, 24, 26–27, 29–35, 38–39, 41, 46, 48 and 50 from IMDb, that is we used 32 topics. We set a maximum retrieval depth of 20 results.

We used a six-processor 3.33 GHz Intel(R) Xeon(R) machine with 96GB of RAM running Java 1.8 and PostgreSQL 9.1. We set a maximum execution time of 1 hours and maximum 15 Gbytes of memory to be used.

We used the following measures for evaluating effectiveness. *Precision at Ten (P@10)* is the classic precision measure with cut-off at the first 10 retrieved documents: $\text{Prec}(n) = \frac{1}{n}\sum_{i=1}^{i} r_i$, where r_i is 1 if a relevant result is found; 0 otherwise. Note that several topics have only 1 relevant document and this affects P@10 which can only be 0.1 in this cases. *Average Precision (AP)* represents the "gold standard" measure in IR, known to be stable and informative, with a natural top-heavy bias and an underlying theoretical basis as approximation of the area under the precision/recall curve: $AP = \frac{1}{RB}\sum_{i\in\mathcal{R}} \text{Prec}(i)$, where $\mathcal{R}$ is the sets of ranks at which relevant results have been retrieved and RB is the recall base, i.e. the total number of relevant documents.

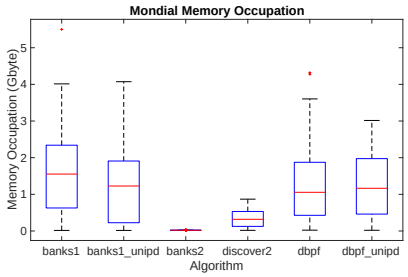
4 Experimental Results

4.1 Efficiency

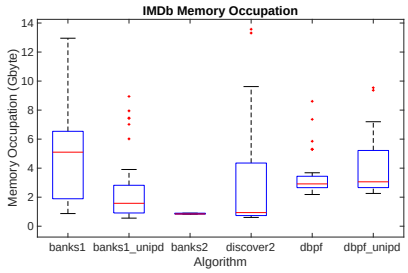
Figure 1 shows the memory occupation and execution times for running the queries: each sub-figure reports a boxplot computed over the different queries. Figure 1(e) reports the mean performance.

Memory-wise BANKS-I and DPBF are the most demanding algorithms on both Mondial and IMDb while their UNIPD variants improve on this, especially in the case of IMDb when there are lots of data and we save between 15% and 40% in memory occupation. DISCOVER-II performs very well on Mondial and better than BANKS-I and DPBF on IMDb. BANKS-II always outperforms all the other algorithms, requiring a very small memory footprint. Finally, Figure 1(a) and 1(b) show quite a bit variance in the memory occupation across the different queries and, especially in the case of IMDb, there are quite high outliers. This indicates that the algorithms are quite affected by the topic and collection at hand and that the performance can vary quite a lot. The most robust algorithm in this respect is BANKS-II, which has a very low variance.

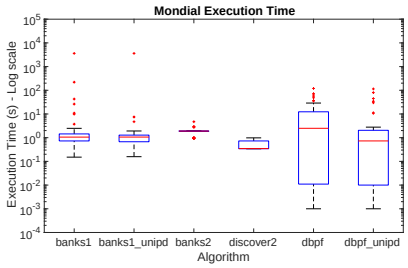
[10] reports in Table 10 at page 38 the memory occupation for creating the graph and the initial memory required by a search technique, so these figures does not seem directly comparable with the memory occupation per query execution



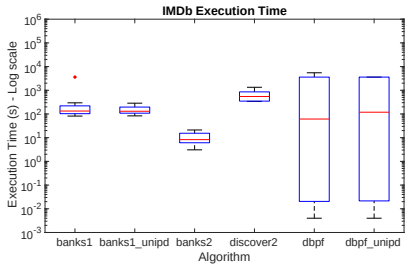
(a) Mondial memory occupation.



(b) IMDb memory occupation.



(c) Mondial execution time.



(d) IMDb execution time.

Algorithm	Mondial		IMDb	
	Memory (Gbyte)	Time (s)	Memory (Gbyte)	Time (s)
BANKS-I	1.597	93.939	4.602	576.835
BANKS-I UNIPD	1.174	86.865	2.630	149.971
BANKS-II	0.020	1.837	0.856	10.423
DISCOVER-II	0.358	0.482	2.976	653.302
DPBF	1.270	14.337	3.481	1,544.736
DPBF UNIPD	1.192	8.355	4.025	1,366.078

(e) Average memory occupation and execution time.

Fig. 1. Efficiency: memory occupation and execution time for running the queries.

we report in Figure 1. However, we may try some very qualitative observation: [10] reports, in the case of Mondial, a memory occupation in the range of tens of megabytes while Figure 1(a) shows a memory occupation in the range of gigabytes, except for BANKS-II and DISCOVER-II; in the case of IMDb, both [10] and Figure 1(b) report a memory occupation in the range of gigabytes, except for BANKS-II in our case. As discussed in Section 2, most of the papers lack any details on the actual data structures to be used and this leads to the differences we observed above.

Time-wise, on Mondial DISCOVER-II is the fastest algorithm followed by BANKS-II, the DPBF variants, and, finally, the BANKS-I variants; on IMDb, BANKS-II is the fastest algorithm, followed by the BANKS-I variants, DISCOVER-II, and DPBF. Figure 1(c) and 1(d) show how, in general, there is less variance

in the execution times than in memory occupation, even if DPBF exhibit the largest spread of performance and BANKS-I suffers from several high outliers in the case of Mondial. We can note how the UNIPD variant of BANKS-I improves around 8% and 75% on Mondial and IMDb, respectively; this indicates that the optimization on the length of the paths is most effective with larger amounts of data, as in the case of IMDb, when we also save about 40% in memory occupation. The UNIPD variant of DPBF performs similarly on IMDb but it saves about 42% of execution time on Mondial; this indicates that the optimization in the merging steps is most effective for more complex schemas, as expected.

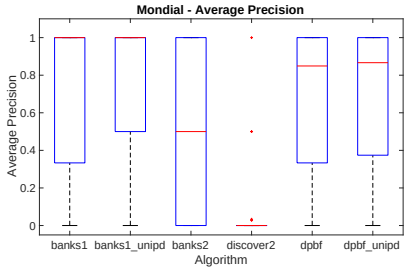
Since we used a different hardware than [10], execution times are not directly comparable in a quantitative way, even if it is possible to discuss trends in a qualitative manner. Tables 8(a) and 8(b) at page 37 of [10] report the mean execution times for Mondial and IMDb, respectively. In the Mondial case, according to [10], DISCOVER-II is the fastest algorithm, followed by DPBF, BANKS-II, and BANKS-I; we also note that [10] reports execution times one/two orders of magnitude bigger than ours. In the case of IMDb, according to [10], DISCOVER-II is the fastest algorithm, followed by BANKS-I, DPBF, and BANKS-II; in this case [10] reports execution times 2-3 times bigger than ours. Overall, there is some agreement in the ranking of algorithms on Mondial but there is quite disagreement in the case of IMDb. As in the case of the memory occupation, we think all these differences in the execution times are due to quite different choices at implementation level.

4.2 Effectiveness

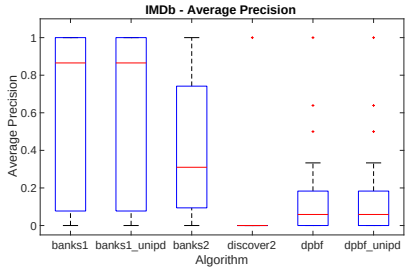
Figure 2 shows the effectiveness of different algorithms under examination in terms of AP and P@10: each sub-figure reports a boxplot computed over the different queries while Figure 2(e) reports the mean performances for AP and P@10 on each collection. Note that, since by construction of the dataset many queries have just one relevant result, in many cases it is not possible to have P@10 bigger than 0.10.

In the case of AP, on Mondial the BANKS-I variants are the top performing algorithms, followed by the DPBF versions, BANKS-II, and DISCOVER-II. In particular, the UNIPD variants of both BANKS-I and DPBF outperform the original versions and BANKS-I UNIPD is the top performer. On IMDb, the BANKS-I versions are the top performing algorithms, followed by BANKS-II, DPBF, and DISCOVER-II; in this case the UNIPD variants perform very similarly to the original algorithms. So we can deduce, that the UNIPD variants of BANKS-I and DPBF provide benefits in terms of memory occupation and/or execution time, as shown in Figure 2, and they keep or even improve the effectiveness in terms of AP.

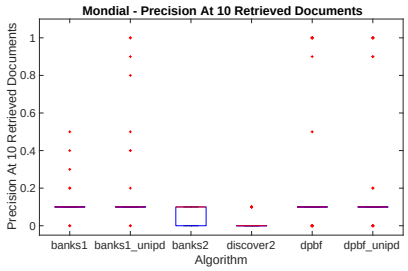
The boxplots of Figures 2(a) and 2(b) indicate quite a lot of variance in the performances and, especially, distributions skewed towards 1.0 AP. This is a very different behaviour with respect to the one typically observed in IR evaluation and it might be more an indicator of overfitting and/or design issues in the dataset rather than high performances of these algorithms.



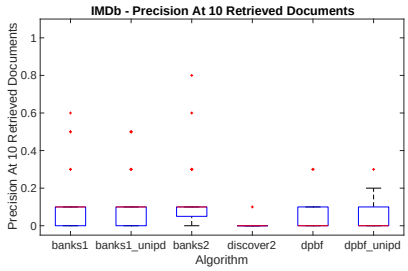
(a) Mondial AP.



(b) IMDb AP.



(c) Mondial P@10.



(d) IMDb P@10.

Algorithm	Mondial		IMDb	
	MAP	Mean P@10	MAP	Mean P@10
BANKS-I	0.7166	0.1214	0.6245	0.1313
BANKS-I UNIPD	0.7553	0.2000	0.6240	0.1281
BANKS-II	0.5220	0.0667	0.4100	0.1375
DISCOVER-II	0.0498	0.0143	0.0312	0.0031
DPBF	0.6575	0.2167	0.1393	0.0500
DPBF UNIPD	0.6699	0.2381	0.1376	0.0469

(e) Mean performances for Mondial and IMDb.

Fig. 2. Efficiency: precision at 10 retrieved documents and average precision.

If we compare Figures 2(a) and 2(b) with Figure 7 at page 40 of [10], we can observe several differences. On Mondial, [10] identifies DPBF as the top algorithm, followed by BANKS-II, DISCOVER-II, and BANKS-I; on IMDb, [10] ranks the algorithms as BANKS-I, DISCOVER-II, DPBF, and no results for BANKS-II. These trends are quite different from those emerging in our experiments. For example, if we look at mean performance of BANKS-I, on Mondial [10] roughly[7] reports 38% of MAP while we have around 71% and, on IMDb, 20% against 62% in our case; we can find similar discrepancies in the case of the other algorithms.

[7] There are no tables in [10] with precise values for MAP and P@10 and we have to "deduce" them from the histograms in their figures.

We think that the cause of these differences is two-fold. Firstly, we used only the topics where all the algorithms retrieved at least one result while [10] does not specify how topics without results are considered and averaged. Secondly, the difficulty in reproducing the results of [10] indicates that the issues we encountered in implementing the various algorithms have led to implementations that are non-identical to those of [10]. However, these differences may be due to our own choices to address the lack of details previously pointed out but also the implementations of [10] may be based on choices and assumptions not explicitly reported which made them different from the original algorithms; so, overall, it is not easy to determine which is the closer implementation of the original algorithms, also because those of [10] are not available online as open source.

When it comes to P@10, on Mondial, the DPBF versions are the top algorithm, followed by the BANKS-I versions, BANKS-II, and DISCOVER-II; also in this case, the UNIPD variants of both BANKS-I and DPBF improve with respect to their original versions and DPBF UNIPD is the top performer. On IMDb, BANKS-II is the top performer, followed by the BANKS-I versions, the DPBF versions, and DISCOVER-II; also in this case the UNIPD variants perform very similarly to the original algorithms.

Also in the case of P@10, if we compare our results with those of [10], we can find several differences and it is hard to reproduce the same results.

5 Discussion and Future Work

In this paper, we have tried to reproduce several state-of-the-art algorithms for keyword-based search over relational databases and compare our results with those of [10], which represents one of the most comprehensive studies so far.

As discussed in Section 2, we found that, in general, the papers describing these algorithms lack many details on both the exact data structures to be used and some specific steps of each algorithm. This lack of details is also confirmed by the diversity of choices made by the student groups, which worked independently: for example, even if they all had to represent a graph, they all adopted a different approach to this problem.

As discussed in Section 4, we obtained experimental results different from those of [10] in terms of both efficiency and effectiveness. So, we discovered that it is extremely hard to reproduce these experiments.

It could be argued that keyword-based search over relational databases is a research topic and master students may be not experienced enough yet to properly deal with it, leading to implementations with bugs or misinterpretations of an algorithm. Our implementation will certainly contain bugs and/or errors, nevertheless a striking question arises: how difficult should be to re-implement keyword-based search over relational databases algorithms? Our master students attended courses on algorithms and data structures, advanced algorithms, databases and, about half of them, information retrieval; so you can guess that they have quite a good mix of competencies to study and implement algorithms whose core is basically navigation in graphs. The fact that they found so difficult,

if not impossible, to reproduce the results suggests that these algorithms are not explained clearly enough and that the barrier to just start working on this topic is very high. Think about a PhD student: how long she/he should take to just get started on the topic? Would one year (or more) be a reasonable amount of time?

Overall, this experience further stresses the issues we discussed in the introduction: keyword-based search over relational databases completely lacks a reference architecture and, as a consequence, also a reference implementation of state-of-the-art algorithms. If these algorithms turn out to be so difficult to implement from scratch, it becomes even more compelling to have reference implementations available, where all the possible care and expertise have been put in correctly implementing them.

Therefore, this work represents a first step towards shared and common implementations of such algorithms. Indeed, notwithstanding all the limitations that a student project may have, we implemented several algorithms which are online available[8] under a very liberal open source license, and everyone is now free to start from, correct, improve and compare against these implementations.

Our future works concerns revising and consolidating all these independent implementations into a single an coherent framework to begin providing a "reference" implementation of keyword-based search algorithms.

References

1. D. Abadi, R. Agrawal, A. Ailamaki, M. Balazinska, P. A. Bernstein, M. J. Carey, S. Chaudhuri, J. Dean, A. Doan, M. J. Franklin, J. Gehrke, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, D. Kossmann, S. Madden, S. Mehrotra, T. Milo, J. F. Naughton, R. Ramakrishnan, V. Markl, C. Olston, B. C. Ooi, C. Rè, D. Suciu, M. Stonebraker, T. Walter, and J. Widom. The Beckman Report on Database Research. *SIGMOD Record*, 43(3):61–70, 2014.
2. A. Badan, L. Benvegnù, M. Biasetton, G. Bonato, A. Brighente, A. Cenzato, P. Ceron, G. Cogato, S. Marchesin, A. Minetto, L. Pellegrina, A. Purpura, R. Simionato, N. Soleti, M. Tessarotto, A. Tonon, F. Vendramin, and N. Ferro. Towards open-source shared implementations of keyword-based access systems to relational data. In *Proc. 1st International Workshop on Keyword-Based Access and Ranking at Scale (KARS 2017) – Proc. of the Workshops of the EDBT/ICDT 2017 Joint Conference (EDBT/ICDT 2017)*. CEUR Workshop Proceedings (CEUR-WS.org), ISSN 1613-0073, `http://ceur-ws.org/Vol-1810/`, 2017.
3. S. Bergamaschi, N. Ferro, F. Guerra, and G. Silvello. Keyword-based Search over Databases: A Roadmap for a Reference Architecture Paired with an Evaluation Framework. *LNCS Transactions on Computational Collective Intelligence (TCCI)*, 9630:1–20, 2016.
4. M. K. Bergman. The Deep Web: Surfacing Hidden Value. *The Journal of Electronic Publishing*, 7(1), August 2001.
5. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *Proc. 18th International*

[8] `https://bitbucket.org/ks-bd-2015-2016/ks-unipd`

Conference on Data Engineering (ICDE 2002), pages 431–440. IEEE Computer Society, Los Alamitos, CA, USA, 2002.
6. M. Bron, K. Balog, and M. de Rijke. Example Based Entity Search in the Web of Data. In *Advances in Information Retrieval. Proc. 35th European Conference on IR Research (ECIR 2013)*, pages 392–403. Lecture Notes in Computer Science (LNCS) 7814, Springer, Heidelberg, Germany, 2013.
7. M. J. Cafarella, A. Halevy, and J. Madhavan. Structured Data on the Web. *Communications of the ACM (CACM)*, 54(2):933–947, February 2011.
8. M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. WebTables: Exploring the power of tables on the Web. *Proceedings of the VLDB Endowment (PVLDB)*, 1:538–549, 2008.
9. C. W. Cleverdon. The Cranfield Tests on Index Languages Devices. *Aslib Proceedings*, 19(6):173–194, 1967.
10. J. Coffman and A. C. Weaver. An Empirical Performance Evaluation of Relational Keyword Search Techniques. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 1(26):30–42, 2014.
11. B. Ding, J. Xu Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding Top-k Min-Cost Connected Trees in Databases. In *Proc. 23rd International Conference on Data Engineering (ICDE 2007)*, pages 836–845. IEEE Computer Society, Los Alamitos, CA, USA, 2007.
12. H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting Relational Tables from Lists on the Web. *Proceedings of the VLDB Endowment (PVLDB)*, 2:1078–1089, 2009.
13. N. Ferro. Reproducibility Challenges in Information Retrieval Evaluation. *ACM Journal of Data and Information Quality (JDIQ)*, 8(2):8:1–8:4, February 2017.
14. N. Ferro, N. Fuhr, K. Järvelin, N. Kando, M. Lippold, and J. Zobel. Increasing Reproducibility in IR: Findings from the Dagstuhl Seminar on "Reproducibility of Data-Oriented Experiments in e-Science". *SIGIR Forum*, 50(1):68–82, June 2016.
15. N. Ferro and G. Silvello. The Road Towards Reproducibility in Science: The Case of Data Citation. In *Proc. 13th Italian Research Conference on Digital Libraries (IRCDL 2017)*. Communications in Computer and Information Science (CCIS), Springer, Heidelberg, Germany, 2017.
16. V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In *Proc. 29th International Conference on Very Large Data Bases (VLDB 2003)*, pages 850–861. Morgan Kaufmann Publisher, Inc., San Francisco, CA, USA, 2003.
17. V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional Expansion For Keyword Search on Graph Databases. In *Proc. 31st International Conference on Very Large Data Bases (VLDB 2005)*, pages 505–516. ACM Press, New York, USA, 2004.
18. H. Wang and C. C. Aggarwal. A Survey of Algorithms for Keyword Search on Graph Data. In *Managing and Mining Graph Data*, pages 249–273. Springer-Verlag, New York, USA, 2010.
19. A. Wright. Searching the Deep Web. *Communications of the ACM (CACM)*, 51(10):14–15, October 2008.
20. W. Wu. Proactive Natural Language Search Engine: Tapping into Structured Data on the Web. In *16th International Conference on Extending Database Technology (EDBT 2013)*, pages 143–148. Academic Press, New York, USA, 2013.
21. J. X. Yu, L. Qin, and L. Chang. *Keyword Search in Databases*. Morgan & Claypool Publishers, USA, 2010.