Toward a framework for graph-based keyword search over relational data

Vittoria Cozza

ENEA,
Casaccia Research Centre,
via Anguillarese, 301-00123 Rome, Italy
Email: vittoria.cozza@enea.it

Abstract: Keyword-based access to structured data has attracted research and industry as a means for facilitating access to information. In recent years, the research community and big data technology vendors put a lot of efforts into developing new proof of concept systems for the task at hand. Two major limitations have been identified for such prototypes to transition into fully developed products: 1) systems are not designed to scale up; 2) the absence of a complete evaluation approach oriented towards effectiveness. This work presents a framework for supporting the development and the evaluation of graph-based keyword search systems. Furthermore, the implementation of a core module of this framework is detailed and shared open-source with the community.

Keywords: relational data; database search; structured data; graph search; keyword search; database applications; knowledge management applications.

Reference to this paper should be made as follows: Cozza, V. (xxxx) 'Toward a framework for graph-based keyword search over relational data', *Int. J. Intelligent Information and Database Systems*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Vittoria Cozza received her MSc in Computer Engineering from the University of Calabria, Italy in 2006 and PhD in Computer Science from University of Bari, Italy in 2010. She holds a Second Level Master in Computer Security and Digital Forensics in 2011 from the University of Modena and Reggio Emilia, Italy. She has worked on data science research projects with several national research institutions and companies. She gained experience teaching computer programming and data management to undergraduate students. Currently, she is a computer engineering researcher at Italian National Agency for New Technologies, Energy and Sustainable Economic Development (ENEA), Rome, Italy. Her research interests include: information retrieval and search engine results personalisation, natural language processing and data privacy.

1 Introduction

The steady growth of structured data freely available to everybody (Cafarella et al., 2011) pushes researchers to find new ways for making the data easily accessible and retrievable (Noy et al., 2019; Chapman et al., 2020). The usage of the Keyword Search (KS) paradigm enables any user to search over structured data, by specifying a set of keywords. No knowledge of formal query languages or awareness of the schema is required.

In the last decades, several keyword search systems (KSS) over relational database (RDB) have been implemented (Yu et al., 2009; Park and Lee, 2011). According to Yu et al. (2009), state-of-the-art KSSs, from now always over RDB, are classified into *graph-based* and *schema-based*. The former category exploits a graph built on a relational database (RDB) instance, the latter the RDB schema. *Graph-based* KSSs convert a RDB into a data-graph (DG) where nodes are the RDB tuples and edges represent the primary/foreign key relations between tuples. The KS is answered by performing the search of relevant subgraph structures over the DG. *Schema-based* approaches imply making use of the RDB schema information to generate a set of SQL queries that can find all the structures among tuples in an RDB completely, and how to evaluate the generated set of SQL queries efficiently.

Reproducibility studies (Coffman and Weaver, 2010a, 2014; Badan et al., 2017b) have analysed the strengths and limitations of existing KSSs. In Coffman and Weaver (2010a), and the follow up work (Coffman and Weaver, 2014), the authors reproduced and evaluated Schema- and Graph-based pioneering KSSs, moreover they shared a benchmark (Coffman and Weaver, 2017) for KSS evaluation. The experimental analysis from Coffman and Weaver (2014) reveals that existing KSSs are quite complex and often do not scale for real size RDB. *Graph-based* KSSs outperform the others for what concerns complex searches in complex structure RDBs with thousands of tuples, while they implement search algorithms failing with out-of-memory issues or running out of time for not trivial searches in larger RDB.

This study has three main limitations. First, search queries proposed in the benchmark are not general enough to challenge the systems and this can lead to a shallow evaluation. A KS with one or more keywords that are unique in the RDB is usually easy to answer. Time and space complexity is limited for algorithms solving such cases. The problem arises when the search keywords are present in several attributes of different relations. Hence, search algorithm needs to stress the system finding all the possible connections between all the RDB elements that matched the keywords; this may imply the generation, and also the evaluation, of several subgraphs. Second, Coffman and Weaver (2014), while reproducing main KSSs, refer to their description in the original papers where several details are often taken for granted and important discussions are omitted, thus, the produced results are not always replicable (Bergamaschi et al., 2016). Lastly, the source code of the tested systems is not shared. Badan et al. (2017b) introduced a first attempt in this direction. Badan et al. (2017b) reproduced six KSSs as student projects and discussed the main challenges faced. Moreover, they evaluated the reproduced systems (Badan et al., 2017a) and also share the source code.

The present work introduces the design of a framework that supports the reproducibility and new implementation of graph-based KSSs. It also presents the description and the implementation (Cozza, 2019) of a core module of this framework,

the *data-graph modelling* component, that transforms a given RDB into a DG representation (component source code link hidden for double blind review). In order to favour the reproducibility of the systems, the analytical model of the DG weights is provided (see Section 4). The reproduced module has been evaluated by using it to generate DGs from three RDBs available in Coffman and Weaver (2010b). These RDBs have been also used in the reproducibility work from Coffman and Weaver (2014), thus, it has been possible to validate the correctness of the proposed implementation, comparing the obtained graph structures with those of Coffman and Weaver (2014).

The rest of the paper is organised as follows: Section 2 presents the design of the Graph-based KSS framework, from Sections 3 to 5 the implementation and the evaluation of the *data-graph modelling* module are detailed and discussed. Section 6 concludes the work.

2 Graph-based KSS framework

The design of the framework is mainly based on the pioneering *graph-based* KSSs: BANKS-I (Bhalotia et al., 2002), BANKS-II (Kacholia et al., 2005) and DPBF (Ding et al., 2007). Several aspects of these systems have been analysed in order to identify the parts in common and their interactions. The framework can easily incorporate other *graph-based* KSSs, e.g., He et al. (2007) and Kasneci et al. (2009).

A core module of the framework is the *data-graph modelling*, this builds a DG representation of the RDB instance (see Section 3): relational tuples are nodes of the graph, tuples primary/foreign key relations are edges; hence, the module assigns weights to the graph elements by using graph-based metrics (see Section 4). Figure 1 shows, as running example, an oversimplified version of an Academy RDB where an employee can be affiliated to one and only one department. The latter can have many employees. Each node is identified by an alphanumeric code that is the code reported at the left side of each tuple in Figure 1. Figure 2 shows the graph obtained through the *data-graph block*, having the sample database as input.

Figure 1 A snapshot of the academy RDB schema with its data

Employee

	id	name	surname	role
E0	1	Alice	White	Researcher
E1	2	Bob	Brown	Professor

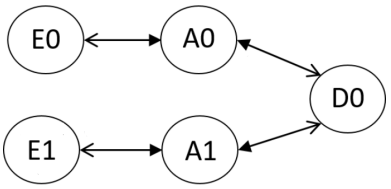
Affiliation

	e_id	d_name
A0	1	DEI
A1	2	DEI

Department

	name	street_name	num
D0	DEI	University Road	100

Figure 2 Academy DG



Analysed systems differ from each-other in relation to the type of graph they handle (oriented or not, weighted or not); systems working on weighted graph may differ on how weights are calculated.

Moreover, an inverted index (Zobel and Moffat, 2006) is generated. In order to build the index, textual information contained in the tuples, attributes and tables name in the RDB is pre-processed (e.g., tokenisation, stop word removal) and keywords are extracted; in the index, each keyword is an index key, the corresponding value is the posting list of graph nodes containing that keyword.

In the running example, node E0 derives from one tuple in the $employee$ table. The textual content associated with this tuple is {Alice, White, researcher} plus its table and table attribute names: {employee, name, surname, role}. This is implemented having E0 in the posting list of each of the keywords in brackets. Three sample elements of the inverted index produced are:

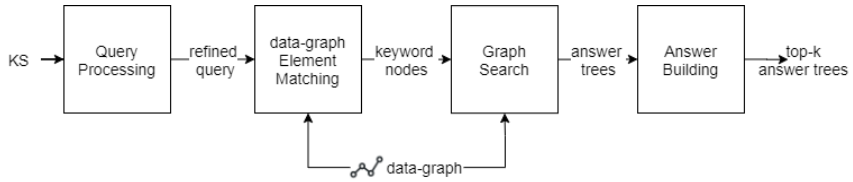
- Alice: {E0}
- researcher: {E0}
- employee: {E0, E1}.

Details on how the graph and the index are built, are provided in the next sections.

Figure 3 represents how the framework answers to a KS, firstly the end user interacts with any KSS by issuing a KS. A KS consists of a set of n terms, named keywords: KS = $\{k_1,...,k_n\}$. The user does not need to be aware of the schema of the data: she is supposed to know only the data domain; thus she may issue a KS by using keywords she expects to be present in any textual attribute in the RDB.

The framework exploits the DG and the KS to produce top-k answer trees. In particular, it runs four modules in pipeline: *query processing*, *data-graph element matching*, *graph search* and *answer building*.

Figure 3 Answering a KS over the data-graph



The first step is the conversion of a KS into a refined query, i.e., a set of keywords, thanks to the *query processing* block. The *query processing* module analyses and processes the KS. At least it must pre-process the KS applying tokenisation and stop word removal, since this pre-processing phase is generally applied to the textual content of the tuples in the RDB. In real systems, NLP techniques can be used in order to understand the semantic of the user query and enrich the search (e.g., query disambiguation and expansion). The *query processing* module returns a refined KS.

In the analysed papers, the *query processing* phase is not discussed or only marginally mentioned, because queries are supposed to be well posed. This shows how far these systems are from being conceived as systems to be used in real life. According

to the running example, given the KS Q_1 = {Bob, at, DEI}, the refined query is Q_1 = {Bob, DEI}.

The *data-graph element matching* block takes the refined KS, and the DG as input along with related indexes. It allows to find all the keywords nodes in the DG, namely the nodes that contain the keywords in KS at least once; this generally is implemented as an access to the inverted index.

According to the running example, given the refined query Q_1, the module returns the following lists of nodes: {E1} for the keyword bob and {A0, A1, D0} for DEI. When a given keyword in KS is not in the inverted index, the module returns for such keyword an empty list of nodes.

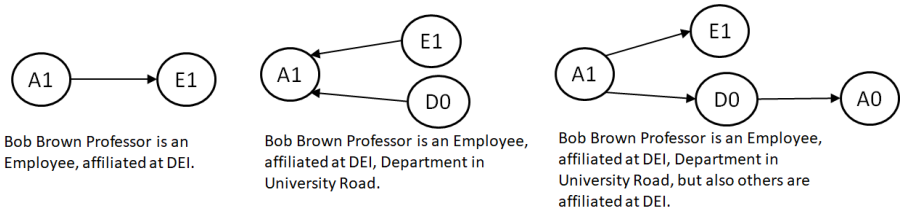
The *graph search* block is used to find connected keyword nodes to form a connected subgraph. This can be done by exploring the neighbourhoods of identified elements and finding the smallest tree connecting all the elements. Even a single node is a subgraph. The module returns the list of valid subgraphs. A subgraph is represented as a list of nodes and its connecting edges, it is valid answer to the KS when its nodes are due to tuples that contain all the keywords in the KS at least once. Indeed in the analysed KSSs, a subgraph answers to a KS when all the keywords in the search are found in its nodes. This behaviour can be extended, e.g., adding in KS Boolean operators and supporting both AND (all the keywords present at least once) and OR semantic (at least one keyword present).

BANKS-I search starts from the keywords nodes 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.

BANKS-I could fail, not producing any result in a reasonable amount of time (Coffman and Weaver, 2014), in case a query keyword matches a very large number of nodes or if it matches a node with a very large number of incoming edges. BANKS-II system improves over BANKS-I the way they traverse the graph by allowing forward search from potential roots towards other keyword nodes. The algorithm uses two concurrent iterators, called outcoming and incoming: both iterators use the Dijkstra's single source shortest path approach but the outcoming iterator expands towards other matching sets using the forward expansion while the incoming iterator implements the same backward expansion strategy of BANKS-I. In addition, BANKS-II favourites expansion of paths with less branching by using a spreading activation mechanism, which assigns an activation score to every explored node, being able to produce results in relevance order. The DPBF system exploits dynamic programming algorithm to find the optimal group Steiner tree. The idea is to find and merge trees with the same root and different sets of keywords until a tree contains all searched keywords is found. In the analysed KSSs, all the subgraph search algorithms proposed are sub-optimal solutions that run in polynomial time aiming to retrieve incrementally relevant results [top-k strategies (Fagin et al., 2001)]. The graph search module returns answer graphs, generally trees, so called *answer trees*, in order of relevance with the user KS. In the example, there are three valid answer trees to the query Q_1. The first produced answer is the tree only made of the nodes A1 and E1 containing the search keywords, and the directed edge (A1, E1) between them. The BANKS algorithm relies on the creation of backward edges for each forward edge, so for each tree there exists also a symmetric

tree, due to the edge in the opposite direction (E1, A1). When the algorithm produces result trees composed of the same nodes, but different edges, only one is returned, based upon relevance. Thus, one other valid result is tree with nodes A1, D0, E1 and edges (E1, A1); (D0, A1); Finally one other more complex result tree is a tree with nodes A0, A1, D0, E1 and edges (A1, E1); (A1, D0); (D0, A0). This tree has a longer path and the analyses systems would assign a higher cost to it too. Intuitively, a longer path means the keyword nodes are loosely related, indeed this tree has a lower relevance then the one mentioned above. Figure 4 summarises the example, along with the intuitively explanation of the results.

Figure 4 Sample answer trees to Q_1 = {Bob, DEI}

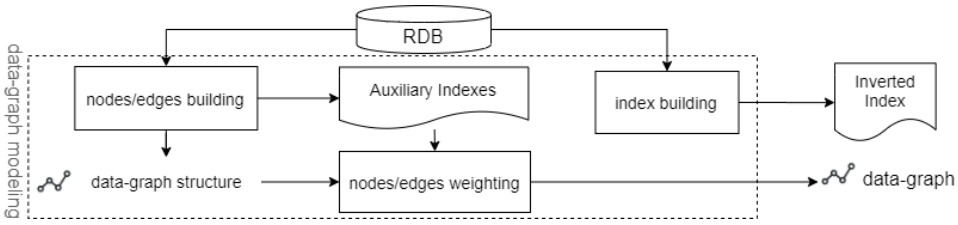


The *answer building* module takes as input the answer trees and it is responsible for assigning them a weight and finally to return the results in relevance order to the user. The relevance of the tree is inversely related to its weight: the higher the relevance the better the answer tree. The tree weight is inherently linked with the prestige of the nodes and the edges proximity. Different KSSs slightly differ on how answer trees are scored, e.g., in BANKS-I and BANKS-II edges and keyword nodes contribute to the tree weight. In the DPBF the tree weight is based on the contribution of the edges only.

3 Building the data-graph

As introduced in Section 2, the *data-graph modelling* block is aimed at building a graph representation of relational data. Figure 5 exemplifies the graph modelling flow: the RDB information is converted into an oriented weighted DG, and the inverted index for indexing textual fields is produced. Furthermore, the module produces and exploits intermediate auxiliary indexes. For instance, an auxiliary structure is a dictionary having as a key the node identifier to store node information (e.g., the relational table name it has originated from), such a data is used to compute edge weight.

Figure 5 Data-graph modelling flow



3.1 Data-graph generation

Formally speaking, given a RDB, upon a relational schema R with foreign key references, the RDB is modelled as a weighted graph G(V, E). V is the set of nodes corresponding to the RDB tuples and E the set of edges induced by foreign key/primary key relationships. This means that for each tuple r in the RDB it must be represented a corresponding graph node u_r. Moreover, for each pair of tuples r_1, r_2 such that there is a foreign key between them, there is a direct and a backward edge between the corresponding nodes u_{r_1} and u_{r_2}.

Figure 2 shows the oriented data-graph from the running example. Open arrows touch forward edges, while closed arrows touch backward edges.

At this point, weights are assigned to the graph edges and eventually to the graph nodes. Especially BANKS-I and BANKS-II systems model a RDB as a directed weighted graph, with forward and backward edges having different weights. DPBF allows to exploit graph oriented and not, weighted and not; in the case of weighted graph, the weight is on the edges only. Weight assignment will be discussed in the next section of this work.

In this phase an inverted index is populated too: for each token in the textual fields of the tuple r in the related posting list the identifier of the node u_r is added. According to the running example, in the inverted index to the keyword DEI corresponds a posting list including the nodes A0, A1, D0. In BANKS-II paper, the authors state that a KS that matches a relation name should select all tuples belonging to the relation. This optional feature was implemented as follows. For each tuple r, while creating the graph node u_r, the node will contain keywords made of the textual content in the tuple r_u, but also the table and table attribute names. For the running example, a posting list including A0 corresponds to the keyword *Alice* in the inverted index. Moreover, while building the inverted index, the identifiers of all the nodes generated by all the tuples in a table are assigned to a keyword corresponding to the table name. For the running example, a posting list including all the nodes due to the table *employee*, specially E0 and E1, corresponds to the keyword *employee* in the inverted index. This solution leads to very long posting list, that can be cumbersome and inflexible for large datasets. In the running example, a posting list including E0, E1 corresponds to the keyword *employee* in the inverted index. In this regard, the authors of BANKS-I mention, as a possible extension to their model, to incorporate queries that specify an attribute name or a table name. E.g., given a query like *role:Researcher*, they expect KSS to retrieve from the inverted index the list of nodes containing the keyword researcher, if and only if the nodes are due to a attribute named role into a RDB table. This feature implementation was left for future work.

3.2 Implementation details

The data-graph modelling block was implemented in Java (Cozza, 2019). It takes as input a PostgreSQL DB instance and it produces a graph representation of the data offline along with the inverted index for full text search (see Figure 5). JDBC interface has been used for managing the connection with a PostgreSQL DB. The module manages any PostgreSQL DB without any extra programming for different RDB schema: the RDB schema is learned at runtime querying the PostgreSQL system catalogues.

The module uses JGraphT library (Michail et al., 2019) for managing graph, also used in He et al. (2007) and Coffman and Weaver (2014). JGraphT is suitable for implementing the whole framework since it supports easy development of graph search code; it provides the implementation of common graph algorithms (e.g., iterators that implement Dijkstra's single source shortest path) either the implementation of very common used functions in KSS search (e.g., $in/out\text{-}degree$ of a node).

The data-graph plus the inverted index are produced as Java serialisable object that can be later loaded in main memory for further elaboration, i.e., for performing graph-search.

However, in the in-memory representation of the graph, the graph nodes are made by the node identifiers only and do not contain actual data. The edge structure stores its two node identifiers and the edge weight. It is well-known that RDBMS implements inverted indexes for full text search. Certain KSSs exploit the RDB index order to implement a tuple-based index for the RDB (e.g., Hristidis and Papakonstantinou, 2002), in the proposed implementation, the inverted index has been made from scratch without relying on RDB indexes. As a limitation, the analysed systems, and therefore the implemented module, do not deal with the case of RDB updates: the module must be run once again when the RDB schema or data are updated. In order to do a search, the graph and the inverted index are loaded within the main memory. While searching for valid answer trees, new in-memory data structures are populated, and these can grow in size, even leading the search to fail with out-of-memory issues. Using dynamic programming, DPBF reduces the memory waste

4 Weighting the data-graph

After DG structure is generated, nodes and edges weights are assigned, accordingly to the analysed KSSs. BANKS-I description paper introduced the metrics, then applied in BANKS-II too. Moreover, the authors in DPBF claim to inherit the same metric definitions for what concerns the edge weight, but they ignore the weight on the node. Given that, the main description of the used metrics into these systems is due to the BANKS-I description paper. However, for the sake of synthesis some of the graph measurements have been described at a general level only. As a contribution, the metrics for computing the node and the edge score described by definitions in the original work, are presented, by providing the corresponding analytical formulas, shedding the light on the details not specified in the original papers. This is important because it makes easy the module implementation by directly translating into code the formulas unambiguously. As for each tuple r in the database, the graph has a corresponding node u_r, then a tuple and the corresponding node are mentioned interchangeably.

The rest of the section presents how the analysed systems compute the node weight, then the similarity between relations, finally the edge weight based on the similarity of the relations in the RDB to which the tuples belong. Moreover, the properties of the edge weight are discussed.

4.1 Node weight

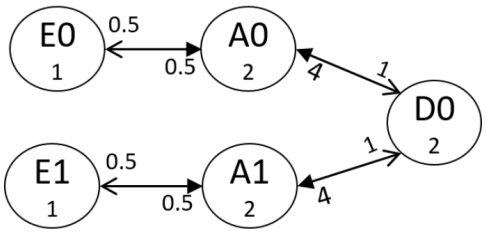
According to Bhalotia et al. (2002), the weight $Nscore$ of a node u is due to the number of its incoming edges:

$$Nscore(u) = IN(u) \tag{1}$$

A higher value of $Nscore$ corresponds to greater prestige for the node in that graph.

For the running example, Figure 6 represents the oriented weighted DG, inside each node the related $Nscore$ is reported. $Nscore(A1) = 2$, having A1 two inner edges, similarly all the other nodes are computed.

Figure 6 Oriented weighted academy DG



4.2 Similarity between relations

As proposed in Bhalotia et al. (2002), given two relations instances R_1 and R_2, where R_1 is the referencing relation and R_2 is the referenced relation, the similarity is a measure based on the type of link between them: when they are unrelated at all similarity is ∞; on the contrary, similarity has to be lower when more semantically relevant is the relation between the connected nodes. Its computation is based on the number of edges between the tuples of the relation instances: $Ne(R_1, R_2)$.

The structure of the graph is based on the RDB instance. The relation type in the relational schema is not considered, but the links between tables are counted and instance-based cardinality is computed accordingly; e.g., the relation actual type is one to many ($\xrightarrow{(1,N)}$) between two given relations R_1 and R_2 when at least one element of R_1 is related to two or more elements of R_2.

The similarity, based on the actual relation types, is formalised as

$$S(R_1, R_2) = \begin{cases} Ne(R_1, R_2) & \text{when } R_1 \xrightarrow{(N,1)} R_2 \\ \frac{1}{Ne(R_1,R_2)} & \text{when } R_1 \xrightarrow{(1,1)} R_2 \text{ or } R_1 \xrightarrow{(1,N)} R_2 \\ \infty & \text{when } R_1 \text{ does not refer to } R_2 \end{cases} \tag{2}$$

In words, the cases are as follows: for a $N \to 1$ relation, similarity is higher when Ne is smaller; for a $1 \to 1$ or a $1 \to N$ relation, similarity is lower when Ne is larger; when $Ne = 0$, the similarity is set to ∞.

Table 1 Actual type of relation, number of edges and similarity for the academy DG

R_1	R_2	*rel type*	Ne	S
$Affiliation$	$Employee$	$1 \to 1$	2	0.5
$Employee$	$Affiliation$	$1 \to 1$	2	0.5
$Affiliation$	$Department$	$N \to 1$	2	2
$Department$	$Affiliation$	$1 \to N$	2	0.5
$Employee$	$Department$	$None$	0	∞
$Department$	$Employee$	$None$	0	∞

Computing the similarity between two tables, following equation (2), requires to compute the number of edges between such tables and the actual relation type. These measurements expect to recall in several computations, so these could be calculated once for all and stored in one auxiliary index. (This optimisation idea is not mentioned in the original paper.)

For the running example, given the tables $Affiliation$ and $Employee$ $Ne(Affiliation, Employee) = 2$, due to the two edges (A0, E0) and (A1, E1); the actual relation type is $1 \to 1$, because each element of $Affiliation$ is related at most to one element of $Employee$.

Hence, according to equation (2) $S(Affiliation, Employee) = \frac{1}{Ne(Affiliation, Employee)} = \frac{1}{2} = 0.5$. For the other couples of tables, the actual relation type, the number of edges Ne, and the similarity S are reported in Table 1.

4.3 Edge score

Given the edge e between the nodes u and v in the graph, being R_u and R_v respectively the relations the nodes u and v belong to, the definition of the weight of e is based on these relations similarity $S(R_u, R_v)$ [see equation (2)] and the number of incoming edges in u or the outgoing edges from v.

For each direct edge, also a backward edge is created; to assign a weight to the backward edge (between v and u), Bhalotia et al. (2002) suggest it can be used a default value or any desired value to reflect the importance of the link between the nodes, with small values corresponding to great proximity. In the proposed implementation, weight for both the backward and the forward edges are computed, this to better measure the nodes proximity. This corresponds to considering the incoming edges in u but also the outgoing edges from v, while in the original formula [equation (1) from Bhalotia et al. (2002)] there was the contribution of the incoming edges only. The link importance is based on how much informative content the connection reveals. As an intuitive example, supposing to have a real size academic RDB, and having two departments named *DEI* and *PHI*, given the *is affiliated to* relation, where the number of edges in *PHI* are less then number of edges in *DEI*, the first links contribute with a more relevant information. Given the opposite direction relation *has affiliated*, the number of edges out from *PHI* are less then number of edges from *DEI*, the first links contribute with a more relevant information.

Following the equations from the original paper and the above considerations, the definition of score of an edge e is formalised as

$$Escore(e) = \begin{cases} IN_v(u) * S(R_u, R_v) & \text{when } R_u \xrightarrow{(N,1)} R_v \\ OUT_u(v) * S(R_u, R_v) & \text{when } R_u \xrightarrow{(1,1)} R_v \text{ or } R_u \xrightarrow{(1,N)} R_v \end{cases} \quad (3)$$

As an example, in order to assign a weight to the edges (E1, A1) and (D0, A1), all the tables which interested nodes, A1, D0, E1 belong to, must be found. Once the related tables are found ($Affiliation$ for A1, $Department$ for D0 and $Employee$ for $E1$), then similarity is computed as discussed in Section 4.2. For the reader convenience, the actual relation type, Ne and S for each couple of tables are evaluated and reported in Table 1.

The weight then relies on the compute of the indegree or outdegree of a node depending on the actual type of relation, according to equation (3).

$$Escore(E1, A1) = IN_{A1}(E1) * S(Employee, Affiliation) = 1 * 0.5 = 0.5$$

having the number of edges incoming to E1 from table affiliation, equals to 1.

$$Escore(A1, E1) = IN_{E1}(A1) * S(Affiliation, Employee) = 1 * 0.5 = 0.5$$

having the number of edges incoming to A1 from table employee, equals to 1.

$$Escore(A1, D0) = IN_{D0}(A1) * S(Affiliation, Department) = 2 * 2 = 4$$
$$Escore(D0, A1) = IN_{A1}(D0) * S(Department, Affiliation) = 2 * 0.5 = 1.$$

Similarly for the other edges. For the reader convenience, the sample graph with edges and nodes score computed as discussed above is reported in Figure 6.

4.4 Edge score proprieties

As a contribution, in the following, the proprieties verified by the edge score are analysed. Table 2 clarifies equation (3) and it shows how the score of the edge between the nodes u and v depends upon the actual type of relation between R_u and v and that between u and R_v.

Table 2 Edge score limit values

rel type	$Escore(e)$
$N \to 1$	$lim_{Ne(R_u,R_v)\to\infty}(IN_u(v) * Ne(R_u, R_v)) = \infty$
$N \to 1$	$lim_{Ne(R_u,R_v)\to 1}(IN_u(v) * Ne(R_u, R_v)) = 1$
$1 \to N$	$lim_{OUT(u)\to 1, Ne(R_u,R_v)\to\infty}(\frac{OUT_u(v)}{Ne(R_u,R_v)}) = 0$
$1 \to N$	$lim_{OUT(u)\to\infty, Ne(R_u,R_v)\to\infty}(\frac{OUT_u(v)}{Ne(R_u,R_v)}) = 1$
$1 \to 1$	$lim_{Ne(R_u,R_v)\to\infty}(\frac{IN_u(v)}{Ne(R_u,R_v)}) = 0$
$1 \to 1$	$lim_{Ne(R_u,R_v)\to 1}(\frac{IN_u(v)}{Ne(R_u,R_v)}) = 1$

Given the edge e, the weight so computed will respect some properties:

- $0 < Escore(e) < \infty$
- $Escore(e)_{1\to1} \leq Escore(e)_{1\to N}$
- $Escore(e)_{1\to N} \leq 1$
- $Escore(e)_{N\to1} \geq 1$.

5 Performance evaluation

In this section the evaluation of the *data-graph modelling* block (see Figure 5) implemented from scratch in this study is presented. The objective of the reproduced block evaluation is two-fold: first, to prove the correctness of the proposed implementation; second, to highlight the open challenges. The module implementation has been tested by using it to convert into a data-graph three real-life RDB, provided by the Coffman and Weaver's (2017) benchmark and that it is possible download for free (Coffman and Weaver, 2010b). This data is interesting because of the different size (see $SIZE$ of the DB in MB and number of relations in Table 3) and structure. Two datasets are extracted from popular websites (IMDb and Wikipedia). IMDB is a very large DB with 1 million tuples, but its structure is quite simple, made by six relations only. Wikipedia has 200k tuples; it is very interesting because it has, in textual fields, all the textual content of Wikipedia articles. It allows to challenge the use of inverted indexes. Finally one dataset is the Mondial dataset (May, 1999), it comprises geographical and demographic information from the CIA World Factbook, the International Atlas, the TERRA database, and other web sources. Mondial has a reasonable small number or tuples (56K) but it is interesting because it counts 28 relations and thus it has a complex structure.

In Coffman and Weaver (2010a, 2014), the authors evaluated several KSSs, using these RDBs. They shared statistics on the produced DGs and the present evaluation refers to part of their study for comparison. The execution time and the memory consumption of the graphs and auxiliary indexes are measured. Besides, the present work also reports how much memory it takes to build them, specially to generate the graph nodes, to create the edges and to update the edge weights.

Table 3 Characteristics of the evaluation datasets and memory consumption

RDB	*Size*	*Relations*	$\|V\|$	$\|E\|$	$\|T\|$	Vm	Em	Em_up	G
Mondial	12	28	17,115	53,748	5,696	43.6 M	20.46 M	2.65 M	10.26 M
IMDb	476	6	1,673,076	6,074,782	279,516	659.92 M	585.54 M	30.08 M	66.05 M
Wikipedia	333	6	206,318	784,600	762,045	1,034.25 M	245.93 M	18.78 M	509.42 M

5.1 Memory consumption

Table 3 shows the characteristics of the RDBs and output DGs: $|V|$ is the number of nodes (tuples) and $|E|$ the number of edges (foreign keys) in DG; $|T|$ is the number

of unique terms (thus the number of inverted index keys). Results are in line with those provided from [Coffman and Weaver, (2014), p.32, Table 3] for what concerns number of relations, nodes and edges, while SIZE and T differ. In Coffman and Weaver (2014), the authors do not explicitly mention how these are calculated. In the proposed experiments, the RDB SIZE has been computed using the pg_DB_size plugin in PostgreSQL, while $|T|$, that represents the number of keys in the inverted index, it depends upon text pre-processing approach and here refers to the usage of a simple rule-based approach.

Furthermore, Table 3 also shows the Data-graph generation module memory consumption while creating the graph, having Vm memory required to generate the nodes and the Index, Em memory required to generate the edges and Em_up to update the edge weight, finally G, the size of the produced graph. Please note that G, the in-memory representation of the RDB, includes only the tuple id identifiers and the edge represented by mean of these identifier, as discussed in Section 3. The memory consumption is the same for the three systems while generating the DG. DPBF system does not consider the node weight, whereas BANKS-I and BANKS-II do but this does not affect the overall memory consumption during graph weighting. Indeed resources necessary to compute the node weight are negligible, this because the weight is based on the in-degree of the node; the in-degree information is pre-computed in the graph representation used in JGraphT, thus in this work, and it comes without a cost. Em and Vm are high because edge and nodes generation are the most memory demanding parts, memory usage could be reduced by improving the implementation, storing some intermediate data in secondary memory. All the analysed KSSs are supposed to update the edge weights (Em_up). This is a very time-demanding task, but it does not waste much memory.

5.2 Execution time

The execution time of the module is biased by the time required for the execution of the SQL queries by the RDBMS underlying the applications (Bergamaschi et al., 2016).

In the proposed experiments, the execution time required to build the DGs from the analysed RDBs is as follows: Mondial needed 6.356 s, IMDb 41 m 48.369 s and Wikipedia 5 m 32.244 s. The reported values refer to an average on three experiments.

Please note that execution time is of around 42 m when the RDB size is of 1 million tuples, but the IMDb version used in Coffman and Weaver (2014), so in the present work, is only a small fraction of the original IMDb RDB (having around 12 million tuples). Since the graph has built offline and loaded in main memory, RDB that requires frequent data or schema updates cannot be easily managed. These, instead, should rely on different approaches that work directly with data stored in secondary memory (see the Schema-based KSSs, as presented in Yu et al. (2009). In this computation, time needed to update the graph edges weight highly contributes to the running time. The results here reported refers to a Mac machine running macOS Sierra 10.12.5, with 2,6 GHz Intel Core i7 processor and with 16 GB of RAM.

6 Final remarks

KSSs have drawn attention in the last years due to the increasing availability of data in structured form and easily accessible even for not technical users. Despite several prototypes have been designed, state of the art implementations are not scalable and time it takes to return relevant search results is far to be instantaneous.

This work proposes the design of a graph-based KSSs framework that aims to be a reference for engineers and technical developers, helping them to develop new KSS, hence it is open to be extended.

The modules designed in the framework encode the main and minimal functionality that graph-based KSS over RDB must support. Possible directions are as follows. The interaction with the user can be improved, e.g., to support searches expressed natural language (Li and Jagadish, 2014), to integrate well known IR and NLP literature techniques such as query disambiguation and expansion (Demidova et al., 2010) or query refinement (Deng et al., 2017; Baid et al., 2010). Applying query understanding techniques is crucial into a real system that aims to be effective when retrieving all the most relevant results.

Several graph-based weighting mechanisms (Yang et al., 2009, 2011) can be included into the framework, beside the examples proposed in the present work. Moreover, graph-based metrics have been conceived to measure the relatedness of two nodes in a graph (Milne and Witten, 2008), or metrics founding on social network analysis that relies on a combination of the relatedness and Katz centrality of a graph node (Hulpus et al., 2015). Most of these metrics can be adapted and applied for the edge score computation task. The *data-graph element matching* module can be enriched to support enhanced query search (e.g., with AND, OR, NOT operator), or to support the matching of numerical value and thus queries that include aggregation operator (Zeng et al., 2016).

Furthermore, this study presents the implementation of a core module of the framework. The module can be extended to support other input data formats, not only RDB. KSSs for the case of other data formats (XML, RDF) were investigated in related works (Bessai-Mechmache and Alimazighi, 2012; Le and Ling, 2016; Dosso and Silvello, 2020).

The module implementation uses a high level language and a well developed library; such decision promotes the clarity and readability of the code, sometimes at the price of its efficiency. The module can be enhanced to implement memory-efficient solutions and managing data-graphs in secondary memory (Dalvi et al., 2008).

The implementation of the rest of the framework is a future work.

Acknowledgements

Most of the work was done while the author was at Department of Information Engineering, University of Padua, Italy.

The research was supported by the Starting Grants Project DAKKAR (DAta benchmarK for Keyword-based Access and Retrieval) promoted by University of Padua and Fondazione Cariparo, Padua (2017–2020).

References

Badan, A. et al. (2017a) 'Keyword-based access to relational data: to reproduce, or to not reproduce?', *Proc. of the 25th Italian Symposium on Advanced Database Systems, Squillace Lido (Catanzaro)*, Italy, 25–29 June, pp.166.

Badan, A. et al. (2017b) 'Towards open-source shared implementations of keyword-based access systems to relational data', *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)*, Vol. 1810, CEUR Workshop Proceedings (CEUR-WS.org), ISSN 1613-0073.

Baid, A., Rae, I., Li, J., Doan, A. and Naughton, J. (2010) 'Toward scalable keyword search over relational data', *Proc. VLDB Endow.*, September, Vol. 3, Nos. 1–2, pp.140–149.

Bergamaschi, S., Ferro, N., Guerra, F. and Silvello, G. (2016) 'Keyword-based search over databases: a roadmap for a reference architecture paired with an evaluation framework', *Transactions on Computational Collective Intelligence XXI*, Vol. 9630, pp.1–20, Springer-Verlag New York, Inc., New York, NY, USA.

Bessai-Mechmache, F-Z. and Alimazighi, Z. (2012) 'Possibilistic model for aggregated search in XML documents', *Int. J. Intell. Inf. Database Syst.*, September, Vol. 6, No. 4, pp.381–404.

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S. and Sudarshan, S. (2002) 'Keyword searching and browsing in databases using banks', *Proc. 18th Int. Conf. on Data Engineering*, pp.431–440.

Cafarella, M.J., Halevy, A. and Madhavan, J. (2011) 'Structured data on the web', *Commun. ACM*, February, Vol. 54, No. 2, pp.72–79.

Chapman, A., Simperl, E., Koesten, L., Konstantinidis, G., Ibáñez, L-D., Kacprzak, E. and Groth, P. (2020) 'Dataset search: a survey', *The VLDB Journal*, Vol. 29, No. 1, pp.932–943.

Coffman, J. and Weaver, A.C. (2010a) 'A framework for evaluating database keyword search strategies', *Proc. of the 19th ACM Int. Conf. on Information and Knowledge Management, CIKM '10*, pp.729–738, ACM, New York, NY, USA.

Coffman, J. and Weaver, A.C. (2010b) *Relational Keyword Search Benchmark* [online] https://joel-coffman.github.io/resources.html#search (accessed 6 July 2021).

Coffman, J. and Weaver, A.C. (2014) 'An empirical performance evaluation of relational keyword search techniques', *IEEE Transactions on Knowledge and Data Engineering*, January, Vol. 26, No. 1, pp.30–42.

Coffman, J. and Weaver, A.C. (2017) *Benchmark for Relational Keyword Search* [online] https://doi.org/10.18130/V3/KEVCF8.

Cozza, V. (2019) *Db2graph Project* [online] https://bitbucket.org/covitti/db2datagraph/ (accessed 6 July 2021).

Dalvi, B.B., Kshirsagar, M. and Sudarshan, S. (2008) 'Keyword search on external memory data graphs', *Proc. VLDB Endow.*, August, Vol. 1, No. 1, pp.1189–1204.

Demidova, E., Fankhauser, P., Zhou, X. and Nejdl, W. (2010) 'DivQ: diversification for keyword search over structured databases', *Proc. of the 33rd Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, SIGIR '10*, pp.331–338, ACM, New York, NY, USA.

Deng, D. et al. (2017) 'The data civilizer system', *The Biennial Conf. on Innovative Data Systems Research, CIDR*.

Ding, B., Yu, J.X., Wang, S., Qin, L., Zhang, X. and Lin, X. (2007) 'Finding top-k min-cost connected trees in databases', *Proc. of the 23rd Int. Conf. on Data Engineering, ICDE 2007*, pp.836–845.

Dosso, D. and Silvello, G. (2020) 'Search text to retrieve graphs: a scalable RDF keyword-based search system', *IEEE Access*, Vol. 8, pp.14089–14111.

Fagin, R., Lotem, A. and Naor, M. (2001) 'Optimal aggregation algorithms for middleware', *Proc. of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '01*, pp.102–113, ACM, New York, NY, USA.

He, H., Wang, H., Yang, J. and Yu, P.S. (2007) 'Blinks: ranked keyword searches on graphs', *Proc. of the 2007 ACM SIGMOD Int. Conf. on Management of Data, SIGMOD '07*, pp.305–316, ACM, New York, NY, USA.

Hristidis, V. and Papakonstantinou, Y. (2002) 'Discover: keyword search in relational databases', *Proc. of VLDB*, pp.670–681, VLDB Endow.

Hulpus, I., Prangnawarat, N. and Hayes, C. (2015) 'Path-based semantic relatedness on linked data and its use to word and entity disambiguation', *International Semantic Web Conference*.

Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R. and Karambelkar, H. (2005) 'Bidirectional expansion for keyword search on graph databases', *Proc. VLDB*, pp.505–516, VLDB Endow.

Kasneci, G., Ramanath, M., Sozio, M., Suchanek, F.M. and Weikum, G. (2009) 'Star: Steiner-tree approximation in relationship graphs', *2009 IEEE 25th International Conference on Data Engineering*, March, pp.868–879.

Le, T.N. and Ling, T.W. (2016) 'Survey on keyword search over XML documents', *SIGMOD Rec.*, Decmeber, Vol. 45, No. 3, pp.17–28.

Li, F. and Jagadish, H.V. (2014) 'Constructing an interactive natural language interface for relational databases', *Proc. VLDB Endow.*, September, Vol. 8, No. 1, pp.73–84.

May, W. (1999) *Information Extraction and Integration with Florid: The Mondial Case Study*, Technical Report 131, Universität Freiburg, Institut für Informatik [online] http://www.dbis.informatik.uni-goettingen.de/Mondial (accessed 6 July 2021).

Michail, D., Kinable, J., Naveh, B. and Sichi, J.V. (2019) *Jgrapht – A Java Library for Graph Data Structures and Algorithms*, arXiv preprint arXiv:1904.08355.

Milne, D. and Witten, I.H. (2008) 'Learning to link with Wikipedia', *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM '08*, pp.509–518, Association for Computing Machinery, New York, NY, USA.

Noy, N., Burgess, M. and Brickley, D. (2019) 'Google dataset search: building a search engine for datasets in an open web ecosystem', *28th Web Conference (WebConf 2019)*.

Park, J. and Lee, S-G. (2011) 'Keyword search in relational databases', *Knowledge and Information Systems*, Vol. 26, No. 2, pp.175–193.

Yang, X., Procopiuc, C.M. and Srivastava, D. (2009) 'Summarizing relational databases', *Proc. VLDB Endow.*, August, Vol. 2, No. 1, pp.634–645.

Yang, X., Procopiuc, C.M. and Srivastava, D. (2011) 'Summary graphs for relational database schemas', *PVLDB*, Vol. 4, No. 11, pp.899–910.

Yu, J., Qin, L. and Chang, L. (2009) *Keyword Search in Databases*, Synthesis Lectures on Data Management, Morgan & Claypool Publishers.

Zeng, Z., Lee, M. and Ling, T.W. (2016) 'Answering keyword queries involving aggregates and GROUPBY on relational databases', *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016*, 15–16 March, Bordeaux, France, pp.161–172.

Zobel, J. and Moffat, A. (2006) 'Inverted files for text search engines', *ACM Comput. Surv.*, July, Vol. 38, No. 2, pp.6–es.