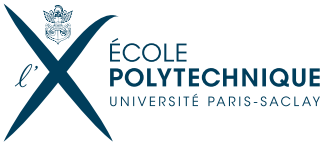
QUERY RESULT SCORES AND QUALITY IN CONNECTIONLENS

Report of 3A's internship

September 2, 2021

Lucas Maia Morais



CONTENTS

1 INTRODUCTION

Investigative journalism is an area of journalism dedicated to deeper and long investigations about a specific subject. This type of investigation is usually more sophisticated to retrieve useful data or and connect different points of interest in the research that aren't clearly related.

The demand for more sophisticated software to help these journalists for finding useful connections in heterogeneous types of data without deep comprehension on how to query information between the data is a valuable resource.

1.1 ConnectionLens [1, 2]

To solve the problem stated above Connection Lens is a project developed by the CEDAR (Rich Data Analytics at Cloud Scale) team, a project team of Inria Saclay and LIX.

Connection Lens is a project that uses graph integration of structured, semi-structured, and unstructured data into a graph database. The system stores and indexes such graphs, and provides a keyword search functionality that helps non-IT specialists to explore the data.

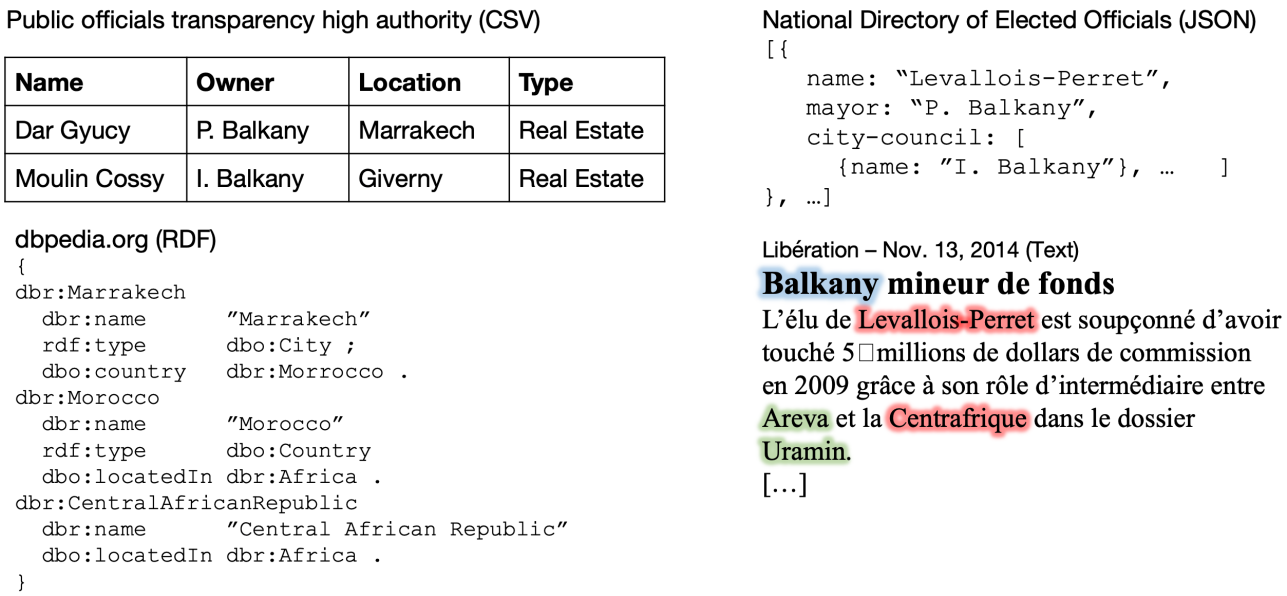


Figure 1: Example of Dataset collection. From [1]

The construction of the integrated graph, e.g. the one from Figure 2, from a given source of data, e.g. the data from Figure 1, in ConnectionLens is detailed in the paper [1]. Here, we briefly explain its main properties.

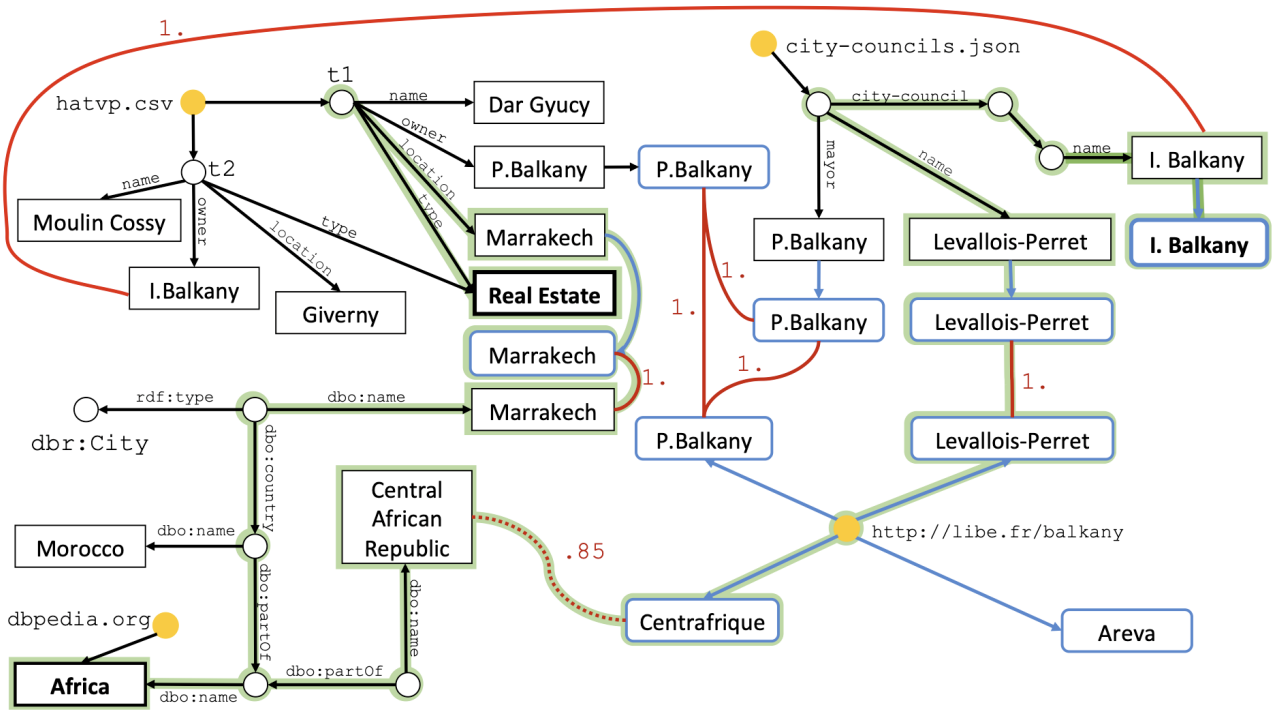


Figure 2: Graph labeled extraction from the datasets shown in Figure 1 (from [1])

There are three types of nodes: *dataset nodes*, for each data source; *struct nodes*, for preserve the internal structure of each data source; and *entity nodes*, represented by rounded blue boxes in Figure 2, that are extracted through dedicated Information Extraction (specifically, Named Entity Recognition) modules, and modeled as children of the text nodes they come from. These NER modules are part of the ConnectionLens system.

Further, a ConnectionLens graph has three classes of edges:

1. *Structural edges*, which connects dataset and structural nodes to structural nodes; these are shown in black in Figure 2;

2. *Equivalence edges*, which connect entity nodes that ConnectionLens considers to be references to the same real-world entity (again see [1] for details, which are orthogonal to the work presented here); these are shown in solid red lines labeled 1.0 (as in: confidence is equal to 1.0) in Figure 2;

3. *Similarity edges*, connecting nodes considered to be similar but not identical, these are shown with dotted red lines in Figure 2.

Further, in Figure 2, a thick green highlight connects the nodes and edges that form a possible answer to the three-keyword query "Balkany, Africa, Real Estate". In ConnectionLens, an answer to this query is a subtree of the graph constructed from all the data sources, such that each keyword is matched by one of the nodes of the tree, and no tree node (or edge) is redundant. We will discuss the search problem in more detail later on in the report, as it is the

main focus of our work. In our example, the matching nodes are labeled "I. Balkany", "Africa", and "Real Estate", respectively (labels shown in bold).

Specifically, after introducing some necessary background in Section 2, we focus on two main features of ConnectionLens search:

- Many queries naturally have several answers on a graph; some queries can actually get a very large number of answers. To help users focus their attention on the most interesting answers, a *score function* is needed. A majority of our work has been on the result scoring problem (Section 3 and 4).
- Because of the very high complexity of the search, and because ConnectionLens makes no assumption on the score function, exhaustive search can be prohibitively expensive - some queries cannot be allowed to run to completion, instead, they are stopped at a time-out, and by that time, they may have produced no results or only very few. The last part of our work (Section 5) introduces a promising *search algorithm optimization* that improves search performance for a large and interesting set of queries.

2 PRELIMINARIES

These are the definitions used for presented in previous papers about the functioning of the ConnectionLens. In the subsections below the information is resumed, for deep comprehension of the details on how the project works, please read the paper [1].

2.1 Graph

We assume a directed, labeled graph $G = (N, E \subseteq N \times N)$ with λ the function assigning labels to node and edges, and c the function assigning some non-negative costs to edges.

2.2 Query

The query in ConnectionLens is designed to be of type keyword search, where a query is made up of query components. These components usually are words, that will pass the entity disambiguation to match similar keywords in the nodes, but the components can also be an exact word, a group of words in a node independent of order, or a group of words in a specific order.

2.3 Algorithmic problems

The comprehension of the algorithmic problem in keyword search in a graph is important to understand the difficulty to solve this kind of problem and why is very important to have a very performing algorithm.

2.3.1 • Steiner Tree Problem

Given a set of nodes $\{n_1, \ldots, n_k\}$, the Steiner tree problem asks for the subtree of G connecting $\{n_1, \ldots, n_k\}$ and having the minimum cost, where the cost is the sum of the costs c of the edges.

2.3.2 • Group Steiner Tree Problem (GST problem)

Given a set of sets of G nodes $\{S_1, S_2, \ldots, S_k\}$, the group Steiner tree problem asks for all the minimum-cost subtrees of G connecting at least one node from each of the sets $S_1, \ldots, S_k$.

According to `http://theory.cs.uni-bonn.de/info5/steinerkompendium/node30.html`, the problem can be approximable within $O(\log^3 n \log k)$ as shown in [33], where n is the number of nodes and k the number of sets, and basically no better solution can be hoped for.

In ConnectionLens, the problem treated is closer to a Group Steiner Tree Problem, where each group corresponds to the set of nodes that match one Query Component.

Remark (2) When we search graphs with keywords, assuming we get a set of keywords $w_1, \ldots, w_k$, and a text index informs us that each keyword w_i appears exactly in the nodes of a set S_i, we are facing a group Steiner tree problem.

Remark (3) In a keyword search setting, there may be also a *matching score*, indicating how well each node matches a keyword. The overall score of a tree may also depend on the matching scores, not just on the edge costs. However, this probably does not change the complexity of the problem.

2.4 Keyword search in ConnectionLens

Two main components need to be described in this setting. The first is the **GAM search algorithm** (Section 2.4.1) that queries the graph to find answer trees. Orthogonal to this component, the **score function** (Section 2.4.2) assigns a score to each tree, which is then used to order the trees.

Importantly, the search algorithm and the score function are fully orthogonal, that is: the score function does not make any assumption on the score function. The advantage is that any score function can be plugged in; the disadvantage is that assumptions on the score function (e.g., assuming that a smaller answer tree is always better) can be used by the search engine to reduce the processing entailed by query evaluation.

2.4.1 • The GAM search algorithm

This algorithm is the one used to search the answers tree (ATs) in the graph labeled. An important remark is that the graph is directed, but the algorithm does a bidirectional search. The algorithm aims at returning *answer trees*, that is, trees consisting of nodes and edges that are part of the original graph, such that each keyword is matched by one tree node. Each such tree has to be *minimal*, in the following sense: (*i*) if one removes an edge from the tree, it becomes disconnected and/or is no longer a solution; (*ii*) no keyword is matched in more than one node (for more details, see [1]).

A simplified explanation of the algorithm can be given using the following three steps:

Initialization A priority queue P of (tree, edge) pairs is created. Then, for each graph node that matches one query keyword, we create a 1-node tree consisting of the respective node; each such tree is then paired successively with every tree edge adjacent to that node, and each such (tree, edge) pair is pushed in the priority queue.

Grow In this step, the (t, e) pair at the top of the priority queue is extracted, where t is a tree rooted in a node denoted r, and e is an edge connecting r to another node. A new tree

t' is constructed, which adds e to t; the root of t' is distinct from the root of t. Thus, all the pairs formed of t' and of an edge adjacent to its root (and which does not close a cycle within t') are added again in the priority queue.

Merge This step is to merge two tree's t_1 and t_2 that have share same root r. Merge leads to the creation of a new, larger tree t''. The algorithm merges *aggressively*, that is: all the trees that could merge with t'' are identified and merged with it, as soon as t'' is built.

Observe that the GAM algorithm starts with nodes matching query keywords; these are leaves in the trees obtained by the first generation of Grow steps. As trees are obtained through Grow and Merge, query keywords are always matched on the leaf nodes. It follows from the notion of minimality introduced above, that in a minimal tree, all the keywords are matched on leaves.

Through Grow, trees get taller (the distance from the root to the leaves increases); through Merge, they get wider (the set of their leaves increases). Thus, eventually, a tree can have a leaf matching each query keyword; at this point, it is a solution.

The algorithm execution ends when k answers tree are found in the case of top k answers, or when all the answers are found, or it runs out of time for a fixed timeout.

2.4.2 • Ranking function

The ranking function is applied to the answers trees found by the search algorithm, to score the answers. There are several score components, and they are combined to create a final score.

1. Matching score $ms(t)$: This is the score related to the similarity of the keywords to the nodes that they match on a tree t. In ConnectionLens, this is the average, over all keywords $w_i \in Q$, where Q is the query, and the node in t that matches w_i using the edit distance.

2. Edge confidence score $c(e)$: This is the score used to calculate the similarity between nodes. It's for example in Figure 2, the edge between the Central African Republic and Centrafrique of 0.85, which was detected to be similar entities by some measure, but as the words inside are not the same, then it's not trivial to say they refer to the same thing. This measure is calculated by the NLP algorithms. This is important because answers that have more uncertainty in their connections, probably are not sure to be the most relevant to the user.

3. Edge specificity $s(e)$: This is a measure to how "rare" is an edge, because answers with low specificity edges or say general are less likely to produce relevant answers. For an edge $e \in E, e = n_1 \xrightarrow{l} n_2$, and a the notation $N^l_{\circ\rightarrow}$ and $N^l_{\rightarrow\circ}$, respectively for the number of edges with label l entering and exiting a node.The specificity of an edge is calculated as follow:

$$s(e) = 2/(N^l_{n_1\rightarrow} + N^l_{\rightarrow n_2})$$

The **final score function** is a linear combination of this three measures for the tree, in the case of ConnectionLens the specificity and confidence of a tree is the multiplication of the contribution of individual edges. So the final measure is expressed by the following formula:

$$score(t, Q) = \alpha \cdot ms(t, Q) + \beta \cdot \prod_{e \in E} c(e) + (1 - \alpha - \beta) \cdot \prod_{e \in E} s(e)$$

Where α and β are parameters to be chosen, such that $\alpha, \beta \in [0, 1)$ and $\alpha + \beta \leq 1$

3 LITERATURE STUDY

The choice of an appropriate result ranking function is of utmost importance for the final user to find as easily as possible the most useful answers among all those found by the query engine.

In this section, we provide an extensive bibliographic review of keyword search algorithms. This is intended to study all possible ranking functions studied until now, on problems related to the keyword search problem studied by ConnectionLens. This study is a contribution to the field of keyword search in heterogeneous sources because it compiles and summarizes previous papers in an organized form.

As we envision publishing this as a survey in the future, given the important volume of works, and the presence in the team of a post-doc (Madhulika Mohanty) with knowledge of some of the works surveyed here, she has contributed initial versions of the sub-sections: 3.1.3, 3.1.15, 3.1.27, and 3.1.28. The remaining ones were studied by myself also with some help from my supervisor.

3.1 Ranking Functions Previously Proposed in the Literature

All the papers reviewed were summarized in four main pieces of information: The type of data ingested by the search engine, the stated keyword search problem or some problem description, the exact definition of an answer to the keyword search problem, and the ranking function used. For ease of reuse of our study, each section is titled with the title of the respective research work, whose year is also recalled. We cover works in their natural, temporal order.

3.1.1 • Keyword searching and browsing in databases using BANKS, 2002 [3]

Type of data: Relational databases. They model a relational database as a graph, where each tuple is a node, and nodes are connected by foreign key and other relationships, such as inclusion dependencies, or some reverse (backward) edges.

Figure 3 shows an example from [3]: a relational database schema (at the top), and a database fragment (at the bottom), where the (primary key, foreign key) edges materialize the graph on which the keyword search applies. A sample query on this graph is "Soumen Sunita", asking for connections between one node matching "Soumen" and another one matching "Sunita".

The authors state that they don't want to view the graph as undirected, because some very well-connected nodes in the undirected graph may lead to small-cost (formally, "good")

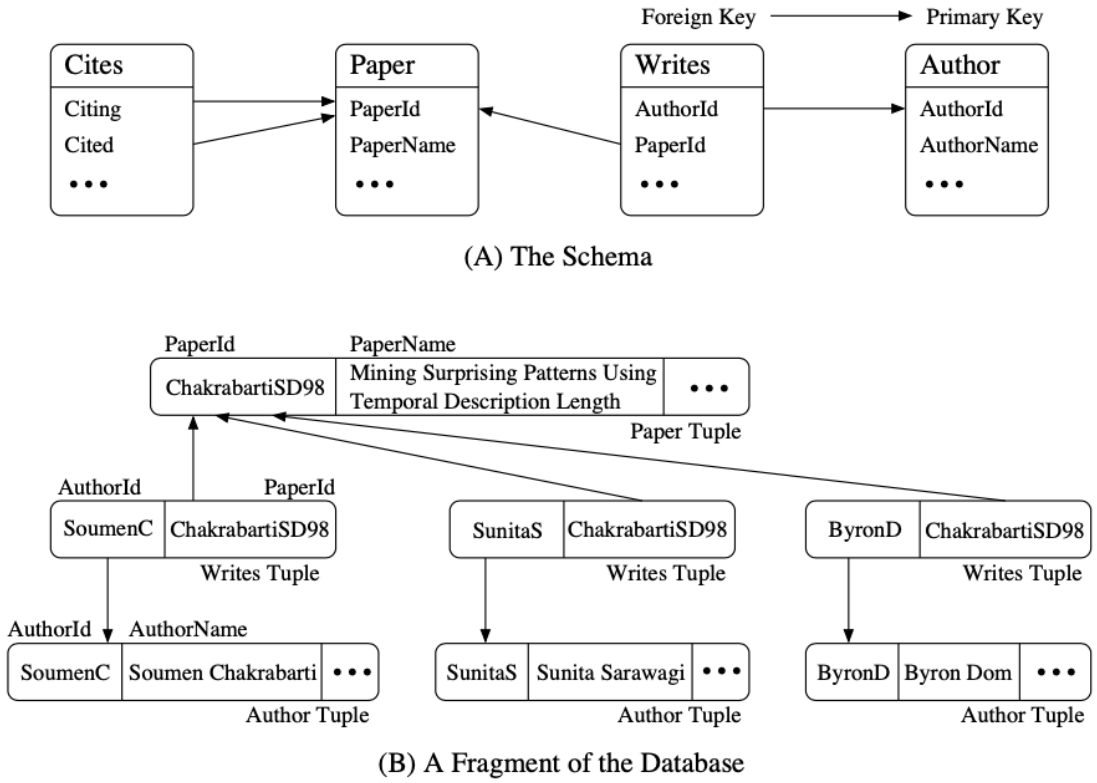


Figure 3: Sample relational database and graph-oriented view of the data (from [3])

solutions, but which are meaningless.
This modeling is shared by all the keyword search works that have considered relational databases. It is implicit whenever the type of data is relational.

Problem: The goal is to answer keyword queries on the graphs thus obtained from relational databases.

Answer: Answers are rooted trees connecting nodes that match the query keywords. A set of useful terms are introduced:

Joined Network of Tuples (JNT) is a set of tuples from the database, such that each of them joins (through a primary key - foreign key pair) with at least another tuple. A JNT is **total (TJNT)** with respect to a given query, if its tuples, together, match all the query keywords.

Joined Tuple Tree (JTT) is a JNT that is also a tree, that is: from each tuple in the JNT, there is only one path reaching any other tuple.

A solution to the keyword query is **TJTT**, that is: a JTT that matches all query keywords.

Ranking function: Answers are ranked using a notion of proximity and prestige of a node using incoming links (similar to web search). Proximity is the distance from a node to another, while prestige is the number of links incoming to the node. Specifically, they introduce:
Node weights: $N(u)$ = in-degree of u
Edge weights for (u, v):

$$w(u,v) = \begin{cases} s(R(u), R(v)) & \text{where } R(u) \text{ is the relation to which } u \text{ belongs, and } s(R(u)) \text{ is a score of this relation, if } (u,v) \text{ exists but not } (v,u) \\ IN_v(u)s(R(v), R(u)) & \text{if } (v,u) \text{ exists but not } (u,v), \text{ where } IN_v(u) \text{ is the in-degree of nodes of type } v \text{ wrt } u \\ min(s(R(u), R(v)), IN_v(u)s(R(v), R(u))) & \text{otherwise} \end{cases}$$

For the relevance score, they defined the following terms:

$$Nscore(v) = N(v)/N_{max} \text{ or } log(1 + N(v)/N_{max})(idf) \text{ for each node } v$$
$$Escore(e) = w(e)/w_{min} \text{ or } log(1 + w(e)/w_{min}) \text{ for each edge } e$$

For an answer tree, they compute an edge score and a node score as follows:

$$Nscore = mean\ of\ Nscore(v) \text{ over the leaves } v$$
$$Escore = 1/(1 + \sum_e Escore(e))$$

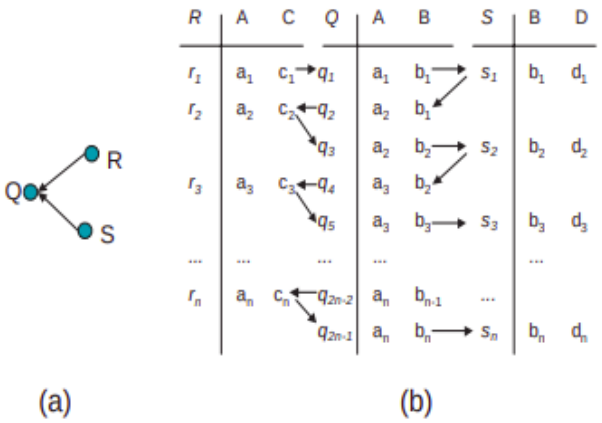


Figure 4: An example of a many-to-many relationship with a JNT bigger than the dataset. In (a) the schema of the example dataset and in (b) a very long JNT for an example of a query searching for "c_1 s_n". (from [4])

They consider two forms of tree relevance scores:

- Additive: $(1 - \lambda) * Nscore + \lambda * Escore$
- Multiplicative: $Escore * Nscore^{\lambda}$

The three options (for edge score, node score, and combination), each of which can take two values, lead to a total of eight combinations. To evaluate these combinations, they check how they score 4 ideal results for a set of 7 queries. They say better results were achieved with **log scaling of edge and** $\lambda = 0.2$.

They note that the score combinations with log scaling nodes or edges and a multiplicative overall score are not useful because scores were too small.

3.1.2 • DISCOVER: keyword search in relational databases, 2002 [4]

Type of data: Relational databases

Problem: The authors make some *assumptions* on the database, such as: no attribute is both FK and PK for two other relations (one attribute can be just PK in one table and FK in another table).

Answer: An answer is a **Minimal Total Joining Network of Tuples (MTJNT)**, that is: a TJTT such that removing a tuple from it makes it disconnected, or makes it miss some of the keywords. As the authors state, *a JNT can be as large as the database*, an example can be seen in Figure 4. As a consequence, they recommend that with the query, users also specify a *maximum size* of the acceptable MTJNTs.

Ranking function: No ranking function is presented since this is not their focus. The authors just want to find all the MTJNT and do this by exploiting the schema as much as possible. The idea is that the schema describes all possible PK-FK joins; therefore, this can be exploited to find "templates" for solutions.

3.1.3 • DBXplorer: A System for Keyword-Based Search over Relational Databases, 2002 [5]

Type of data: Relational databases

Problem: Keyword search over relational databases

Answer: The answers are TJTTs.

Ranking function: The retrieved rows (results of SQL, interpreted as JTTs) are ranked by the number of joins involved (ties are broken arbitrarily); this is equivalent to a ranking by the number of edges (the fewer, the better). The reasoning provided by the authors is that joins involving many tables are harder to comprehend.

3.1.4 • XRANK: Ranked Keyword Search over XML Documents (Guo et al.), 2003 [6]

Type of data: XML documents

Problem: They search in XML documents. The hyperlinks are not considered as edges, just as a measure of awareness (importance) of a node.

Answer: The result is the set of elements that are the *lowest common ancestors* of at least one occurrence of all of the query keywords. That is: for any combination of nodes (one that matches each keyword), their lowest common ancestor is an answer.

Note that here, unusually, an answer is a *node*, not a tree. This is because each answer is seen as "encapsulating" the tree(s) that it is rooted into.

Ranking function: The authors list some desired properties for ranking:

1. Result Specificity: Favor more specific results, e.g., subsections should be ranked better than sections.

2. Keyword proximity: "Close-by" results should rank higher. The authors distinguish proximity (as edit distance in a file) from specificity (which also takes into account the level in the tree).

3. Hyperlink Awareness: The authors state that the number of ID-IDREF links incoming to a node could be leveraged to give a higher score when this number is high.

Given a query $Q = (k_1, k_2, ..., k_n)$, whose result is denoted R, the ranking score they attach to an element v is computed as follows. Let $0 < decay \leq 1$ be a given parameter that degrades scores based on the distance from the root node v.

- Let $(v, v_2)...(v_t, v_{t+1})$ be the path in R from v to the node v_{t+1} matching the keyword k_i. They define $r(v, k_i) = ElemRank(v_t) * decay^{t-1}$, where $ElemRank$ is this the PageRank score of v within the graph.
- If a keyword k_i is matched more than once in R, let $r_1, \ldots, r_m$ be the score computed as above for each of the individual match; they define $\hat{r}(v, k_i) = f(r_1, r_2, .., r_m)$, where f is an aggregation function (by default, f is max)
- Finally, they compute $R(v, Q) = p(v, k_1, \ldots, k_n) \cdot \sum_{1 \leq i \leq n} \hat{r}(v, k_i)$, where p is a proximity function, defined as the size of the smallest text window in v that contains all the query keywords.

The authors suggest that the proximity function in highly structured datasets can be set to 1.

3.1.5 • Keyword Proximity Search on XML Graphs, 2003 [7]

Type of data: XML documents.

Problem: Edges can be followed in both directions. They accept graphs with multiple roots, since:

- They want to support different XML documents in the same graph;
- They prefer that roots of large collections are not included to avoid false proximities.

Answer: The answer of a keyword query is an MTTON (minimal total target object network), which is a tree if one considers the graph undirected.
What to show to users The authors distinguish a query answer (in a computational sense), and *what is shown to the users*. Specifically:

1. They (i) assume the DB admin associates a minimal piece of information, called target object, to each node, and (ii) display the target objects instead of the nodes in the results.
2. They identify a problem of "multiplication" of answers as follows. For a query of 3 keywords a, b, c, there can be nodes a_1, a_2 matching a, connected to a node b_1 matching b, which is connected to two nodes c_1, c_2 matching c. This would lead to four solutions,

based, respectively, on a_1, b_1, c_1, a_1, b_1, c_2, a_2, b_1, c_1 and a_2, b_1, c_2. They say it is wasteful to show four such solutions. They handle this by a combination of steps, in particular showing results on demand and grouping them by their Candidate Network (CN). A Candidate Network is a **schema-level** answer, i.e., it is the image (in the schema graph) of one or many answers.

Ranking function: It seems that the score of a solution is the size in the number of edges for the MTTN (minimal total node network) j equivalent of the MTTON (minimal total target object network) t (which is no guarantee to be unique). The definition that maps the XML graph (where MTTON belongs) to the schema graph (where MTTN belongs) is shown in detail on the paper.

3.1.6 • Efficient IR-style keyword search in relational databases, 2003 [8]

Type of data: Relational databases.

Problem: They consider keyword queries, not only with the standard AND semantics presented so far (all the keywords must be matched) but also with OR semantics (some keywords are optional). The key contribution in this paper is the incorporation of IR information retrieval-style relevance ranking of tuple trees into their query processing framework.

Answer: An answer is a joining tree of tuples (JTT).

Ranking function: The score assigned to a joining tree of tuples consider individual IR-style relevance score for textual attributes implemented in RDBMS and a combination of several other score components.

Let a_i be a textual attribute that is part of an answer tree T. To evaluate the score of a_i with respect to a given query Q, the authors propose to rely on the function below, borrowed from the IR literature [34]:

$$Score(a_i, Q) = \sum_{w \in Q \cap a_i} \frac{1 + ln(1 + ln(tf))}{(1 - s) + s\frac{dl}{avdl}} * ln(\frac{N + 1}{df}),$$

where tf is the frequency of w in a_i, df is the number of tuples in a_i's relation with word w in this attribute, dl is the size of a_i in characters, $avdl$ is the average attribute-value size, N is the number of tuples in a_is relation, and s is a constant (they propose 0.2).

Based on the above, the score of a joining tree of tuples T is defined as

$$Combine(Score(A, Q), size(T)) = \frac{\sum_{a_i \in A} Score(a_i, Q)}{size(T)},$$

where A $= \langle a_1, ..., a_n \rangle$ is a vector with all the textual relational attributes present in T.

3.1.7 • ObjectRank: Authority-Based Keyword Search on Databases, 2004 [9]

Type of data: They search in databases (which they state can come from XML, or relational, or the web)

Problem: They search in a database (which they state can come from XML, or relational, or the web) modeled as a labeled, directed graph.

- They assume available a **schema graph**.
- From the schema graph, they create an **authority transfer schema graph (ATSG)** that reflects the authority transfer rates that each "kind of" edge has. Further, for one edge in the schema graph $u \to v$, there can be different authority transfer rates for $u \to v$ and for $v \to u$, respectively: *"Notice that the amount of authority flow from, say, paper to cited paper or from paper to author or from author to paper, is **arbitrarily set by a domain expert and reflects the semantics of the domain**. For example, common sense says that **in the bibliography domain a paper obtains very little authority (or even none) by referring to authoritative papers**. On the contrary, it obtains a lot of authority by being referred by authoritative papers."*
- From the above, they create an **authority transfer data graph (ATDG)** where each data edge $u \to v$ has an authority transfer based on what it had in the ATSG but also impacted by the number of outgoing edges of u in the graph. Further, the sum of the outgoing authority transfers of u in the ATDG may be smaller than the outgoing transfer corresponding to u in the ATSG, if u does not have all the outgoing properties of its schema counterpart.

Answer: An answer to the query is a ranked **list of nodes** from the database, such that the node leads to query matches. In other words, they only include in the answer *roots* of TTJTs, and if a node is a root to two such trees, they only include it once.

They support both AND semantics (the answer/root node must lead to a match for each keyword) and OR semantics (the answer/root node must lead to a match for at least one of the keywords)[1].

The authors point out that their method may return a node for a 1-keyword query even if the node does not match that keyword! This is an effect of the PageRank-style scoring (see below).

Ranking function: The score of a node v with respect to one keyword w is a combination of

[1] The paper is somehow unclear because on one hand what looks natural is for the root to have edges that lead to the nodes matching the keywords, but on the other hand, when defining the score (see below), they mention the opposite probability, that starting from the nodes matching the keywords, one would reach the root.

the global object rank of v in G (this is query-independent) and a keyword-specific object rank $r^w(v)$.

1. The score of a node v with respect to **one** keyword w is a combination of the global object rank of v in G (this is query-independent) and a keyword-specific object rank $r^w(v)$. The latter reflects the probability that a random surfer, starting from any of the nodes that matched keyword w (these are called $S(w)$), reaches v. In turn, the keyword-specific object rank is computed by solving a fixpoint computation very similar to PageRank, that starts from $S(w)$ and iteratively transfers authority based on the ATSG, until stationarity.

2. The score of a node v wrt **several** keywords is computed as:

   (a) For AND semantics (all keywords must be reachable from the root), as the product of the keyword-wise PageRanks

   (b) For OR semantics, in a more permissive way, starting from $r^{w_1 \vee w_2}(v) = r^{w_1}(v) + r^{w_2}(v) - r^{w_1}(v)r^{w_2}(v)$ for 2 keywords.

They mentioned that one could assign higher weights to the relevance of a node with respect to an *infrequent* keyword.

3.1.8 • BANKS2: Bidirectional Expansion for Keyword Search on Graph Databases 2005 [10]

Type of data: As in BANKS (Section 3.1.1), they search in relational databases,

Problem: As in BANKS (Section 3.1.1), they search in relational databases, viewed as *directed* graphs. They re-state that it is important to view a graph as directed (also) because the strength of the connection between two nodes is not necessarily symmetric. Therefore, for every "forward" edge $u \rightarrow v$ (with weight 1), they *add the opposite-direction edge* $v \rightarrow u$ with weight $w_{vu} = w_{uv} * log_2(1 + indegree(v))$.

Answer: The algorithm looks for approximate solutions to the GST problem. BANKS-2 extends BANKS by allowing *bidirectional exploration* of the answer tree, by exploring what the algorithm considers as promising roots. To decide which node should be forward-explored, BANKS-2 uses a notion of "activation", a heuristic score attributed to candidate nodes.

Ranking function: In a way very similar to BANKS, this article defines a default for the experimentation with the following considerations:

1. The score $s(T, t_i)$ of a tree T with respect to a keyword t_i is the sum of edges weights on the path from the root of T to the leaf containing t_i.

2. The aggregated score of an answer tree T is:

$$E(T) = \sum_i s(T, t_i)$$

3. The prestige of each node is a biased version of PageRank, where the probability of following an edge is inversely proportional to edge weight from the data graph.

4. The tree node prestige score is the sum of the prestige of its leaf nodes and root.

5. The overall tree score is defined as EN^{λ}, where λ adjusts the importance of the edges (they propose a default of $\lambda = 0.2$).

3.1.9 • Keyword Proximity Search in XML Trees, 2006 [11]

Type of data: They search in XML trees.

Problem: They consider keyword search queries, where a query consists of several keywords.

Answer: The answer is the set of minimum connecting trees (MCTs) of the matches of individual keywords.

Ranking function: The quality is not the focus of this paper; the correct results are uniquely defined and for a given set of keyword matches, there is only one[2].

3.1.10 • Effective keyword search in relational databases, 2006 [12]

Type of data: They search in relational databases.

Problem: Queries have OR semantics. There is a (very) strong focus on porting IR-style metrics to a relational database setting.

Answer: An answer is a tuple tree. They use the schema graph to join the nodes. The size of the tuple tree is the number of tuples in it.

Ranking function: The authors make a lot of effort to port IR-style ranking techniques to this relational joining tuple tree setting. Their answer score is a measure of similarity between the query and the tuple tree, such as:

$$Sim(Q, T) = \sum_{k \in Q,T} weight(k, Q) \cdot weight(k, T)$$

[2]The novelty with respect to the work "Keyword Proximity Search on XML Graphs" by the same except Srivastava (XKeyword, Section 3.1.5) is that here they also show how to find the minimum connected trees in *one pass* over the XML tree.

where $weight(k, Q)$ is the term's raw **qtf** (term frequency in the query). For the weight in the tuple tree T, they use four normalizations to adapt tf-idf. Here T is a super-document, and each text column D_i is a document. To compute the weight of T:

$$ntf = 1 + ln(1 + ln(tf)), \text{ where } tf \text{ is the term frequency}$$
$$weight(k, D_i) = \frac{ntf \cdot idf^g}{ndl \cdot Nsize(T)}$$

In the above, ntf has already been defined; the other ingredients are so-called *normalization factors*, which will be defined below. Moreover, they compute

$$weight(k, T) = Comb(weight(k, D_1), \ldots, weight(k, D_m))$$

where $Comb$ combines the term weight into documents, into a term weight into a tuple tree. In detail, the four normalizations they defined are:

Tuple tree normalization ($Nsize(T)$) Bigger trees have more chance to include query keywords. However, simply weighting trees by the raw $size(T)$ suboptimal too, an example is adressed on the paper section *Tuple Tree Size Normalization*. This is calculated as follow:

$$Nsize(T) = (1 - s) + s * \frac{size(T)}{avgsize}$$

where $avgsize$ is the average size of answers.

Document Length normalization (ndl) Different text columns have different sizes, thus keyword matched in columns with fewer words are more important than columns with more words. Good examples of this last consideration appear on the paper section *Document Length Normalization Reconsidered*. The formula they use to reflect this is:

$$ndl = ((1 - s) + s * \frac{dl}{avgdl}) * (1 + ln(avgdl))$$

where dl is the document length in the text column D_i, and $avgdl$ is document length in this column.

Document Frequency Normalization (idf^g) This considers the distribution of words in different columns since a word can appear many times in one column, while in another could be quite rare. To solve this problem, the idea was to use the global values, and not the ones from the text column, thus:

$$idf^g = ln\frac{N^g}{df^g + 1}$$

where N^g are the number of all text columns and df^g is the term frequency in all these text columns.

Inter-Document Weight Normalizations ($Comb()$) This is concerned with the aggregation of terem weights in all documents in T, into a set of term weights in T itself. To account for the fact of bigger answers tree have more terms, the authors propose the formula below inspired from the ntf (term frequency normalization):

$$Comb() = maxWgt * (1 + ln(1 + ln(\frac{SumWgt}{maxWgt})))$$

where $maxWgt$ is the max $weight(k, D_i)$ for a tree T, and $SumWgt$ the sum of the weights.

The authors also discuss the case when a keyword matches a text column name or another schema term. For this case, to adapt the formulas used before, the weight of the keyword with the column name will be adapted to receive a schema-based document frequency. Then the tf will be 1 and the df^g will be the schema global document frequency. Then if an answer T has a tuple that matches both in the text column value and text column name a term, the bigger value between them is used, and the keyword is interpreted as a schema term or a value term.

Phrase search The authors also consider that users may want to search for a given phrase, i.e., certain words appearing in a certain order. They also provide an adaptation of their formulas to this scenario.

3.1.11 • Efficient Keyword Search Across Heterogeneous Relational Databases [13]

Type of data: They search in *a set of* relational databases.

Problem: They consider several, independent databases, which they try to integrate by means of keyword search. Thus, a query is a set of keywords, each of which may match a data node from distinct databases. To connect the matches, the system must identify joins among columns from different databases. In contrast, when searching in a single database, tuples can be interconnected in an answer only by primary key–foreign key joins.

Answer: An answer is a joined tree of tuples, and these tuples can come from different databases.

Ranking function: let T be an answer to Q. Let $a_1, \ldots, a_n$ be relational tuple attributes occurring in T , and $j_1, \ldots, j_m$ be the FK joins used to build T . Furthermore, let $d_1, \ldots, d_m$ be the attribute value pairs "matched" in joins $j_1, \ldots, j_m$, respectively. The score $score(T, Q)$ is defined as: $\frac{\alpha_w \cdot score_w(T,Q)+\alpha_j \cdot score_j(T)+\alpha_d \cdot score_d(T)}{size(T)}$ where α_w, α_j, and α_d are coefficients, and $size(T)$ is the number of joins in T. Further:

- $score_w(T, Q) = \sum_{a_i} score(a_i, Q)$ where $score(a_i, Q)$ quantifies how well a_i matches the keywords in Q; this score is computed using a TF-IDF metric.
- $score_j(T) = \sum_{j_i} score(j_i)$ where $score(j_i)$ measures the confidence in the join j_i that is part of T: if the join is between tables in the same database, the join confidence is 1, otherwise, it is computed based on the system's confidence that the respective *columns* should be joined. Observe that this coefficient is at the level of the schema (it refers to a pair of columns).
- $score_d(T) = \sum_{d_i} score(d_i)$, where $score(d_i)$ is the confidence in joining two *values* from two columns. This part of the score measures the instance-level confidence in a value join.

The authors recommend setting α_w, α_j, and α_d either all identical or to different values, possibly based on user input.

3.1.12 • Spark: Top-k keyword query in relational databases, 2007b [14]

Type of data: They search in relational databases.

Problem: Queries can support both AND and OR semantics and modulate somehow between them (see "Completeness factor" below). The goal is to find the k highest-ranked results. They accept an answer in which a keyword is matched in more than one node (tuple); for a 2-keyword query (a, b), they also accept a 2-tuple answer such that the first tuple matches a and b and the second tuple matches b! (minimality is not a concern).

Answer: An answer is a joined tuple tree (JTT); its size is the number of tuples. A JTT is treated as a **virtual document**.

Ranking function: They propose a score method which they say is better than previous ones, the observe that **is not monotonic with respect to any of its components**.

Each JTT belongs to the result produced by a certain relational algebra expression; the latter is called **Candidate Network (CN)**. They denote by $CN(T)$ the candidate network of a given result tuple T.

Their main score function is:

$$score_a(T, Q) = \sum_{w \in T \cap Q} \frac{1 + ln(1 + ln(tf_w(T)))}{(1 - s) + s \cdot \frac{dl_T}{avdl_{CN^*(T)}}} \cdot ln(idf_w)$$

$$\text{where } tf_w(T) = \sum_{t \in T} tf_w(t),\ idf_w = \frac{N_{CN^*(T)} + 1}{df_w(CN^*(T))}$$

$CN^*(T)$ is identical to $CN(T)$ except that all selection conditions are removed. $CN^*(T)$ is also written CN^* if there is no ambiguity.

This formulas has some problems since calculating $df_w(CN^*)$ and N_{CN^*} would take too much time. Then an approximation is made using $p = \frac{df_w(CN^*)}{N_{CN^*}}$, then the approximation is made considering independence between relations and they estimate p as:

$$\frac{df_w(CN^*)}{N_{CN^*}+1} \approx \frac{df_w(CN^*)}{N_{CN^*}} = p \approx 1 - \prod_j (1 - p_w(R_j))$$

Another term difficult to compute is $avdl_{CN^*(T)}$; they estimate it as $\sum_j avdl_{R_j}$.

This approximation attained an acceptable accuracy (30% relative error) for small CNs (of size up to 3). They claim they will study better approximations in future work!

Other ranking factors They've used some other factors to the score:

- **Completeness factor**: Motivated by the fact that matching different words is better than matching the same word, they map a document into a point in m-dimensional space (where m is the number of keywords in the query), calculate the L^p distance to a ideal document $P_{ideal} = [1, \ldots, 1]$ and normalize it:

$$score_b(T, Q) = 1 - \left(\frac{\sum_{1 \le i \le m} (1 - T.i)^p}{m}\right)^{\frac{1}{p}}$$

  where $T.i$ is the normalized term frequency of a JTT with respect to the keyword w_i, i.e.,

$$T.i = \frac{tf(w_i)}{\max_{1 \le j \le m} tf_{w_j}(T)} * \frac{idf_{w_i}}{\max_{1 \le j \le m} idf_{w_j}(T)}$$

  The p parameter helps to controle the balance between AND and OR semantics, for example $p \to \infty$, then $score_b(T, Q) = min_i(T.i)$, thus, if a JTT doesn't match all keywords, its score is 0.

- **Size Normalization Factor**: A larger JTT (or, equivalently, CN) means more occurrences of keywords. To account for this, a normalization factor is:

$$score_c = (1 - s_1 - s_1 * size(CN)) * (1 + s_2 - s_2 * size(CN^n f))$$

  where $size(CN^n f)$ is the number of non-free tuples (that match keywords) set for the CN. In the experiments they found $s_1 = 0.15$ and $s_2 = \frac{1}{|Q|+1}$ have yielded better results for the queries.

Final Score: The final score takes multiplies the previous factors, i.e.:

$$score(T, Q) = score_a(T, Q) \cdot score_b(T, Q) \cdot score_c(T, Q)$$

3.1.13 • BLINKS: ranked keyword searches on graphs, 2007 [15]

Type of data: They search on directed graphs.

Problem: Keyword search on directed graphs.

Answer: An answer to a query $q = (w_1, ..., w_m)$ is a pair $T = \langle r, (n_1, ..., n_m)\rangle$ where r and w_i are nodes such that: (i) every node n_i contains a keyword w_i and (ii) For every i, there is directed path from r to w_i. The answer is a subtree of the graph, r is the root of them match and n_i the matches. **This only considers edges in the forward direction.**

They formulate a top-k search problem as follows: given a scoring function S, the score of a node r is defined as the score of the best answer rooted in r; the top-k query returns the k nodes with the highest score thus defined, and the best answers rooted at that nodes.

Ranking function: They state that scoring is not the main focus of their work. The scoring function they use is below:

$$S(T) = f(\bar{S}_r(r) + \sum_{i=1}^{m} \bar{S}_n(n_i, w_i) + \sum_{i=1}^{m} \bar{S}_p(r, n_i))$$

where $\bar{S}_p(r, n_i)$ is the shortest-path distance from r to n_i based on a non-negative graph distance measure. Among the terms summed in the input of f, the first reflects a contribution from the answer root, the second from the quality of the matches, and the third from the paths from the root to each match.

They discuss two interesting properties of the score function:

- **Match distributivity**: "the net contribution of matches and root-match paths to the final score can be computed in a distributive manner by summing over all matches. Consequently, all root-match paths contribute independently to the final score, even if these paths may share some common edges". They contrast this with other score functions where "each edge weight is counted only once, even if the edge participates in multiple root-match paths".
- **Reliance on shortest paths**: "the score contribution of a root-match path is the shortest-path distance from the root to the match in the data graph [...] This semantics [...] is intuitive and clean, and allows us to reduce part of the keyword search problem to the classic shortest-path problem. Most of our algorithms and data structures assume these semantics".

In paper section *Optimizations and Other Issues*, they revisit the score issue, providing the following concrete proposal:

$$S(T) = (\alpha * \bar{S}_r(r) + \beta * \sum_{i=1}^{m} \bar{S}_n(n_i, w_i) + \gamma * \sum_{i=1}^{m} \bar{S}_p(r, n_i))^{-1}$$

for some weights α, β, γ that they don't even provide in the experiments.

3.1.14 • Finding top-k min-cost connected trees in databases (DPBF), 2007 [16]

Type of data: Weighted, **undirected** graphs.

Problem: Find the top-1 min-cost connected tree. The authors have sketched an extension for top-k solutions.

Answer: Given a set of keywords, they want to find trees that match all the keywords in their leaves. The cost of a tree is the sum of the costs (weights) of its edges. The answer to the query is then the best (single) tree connecting nodes that match the respective keywords. Only the best (returned) tree is guaranteed to be optimal. The authors sketch an extension to allow it to output *more* trees, but these are not guaranteed to be *the best* result trees.

Ranking function: No meaningful quality score applies, since there is only 1 answer.

3.1.15 • LABRADOR: Efficiently publishing relational databases on the web by using keyword-based query interfaces, 2007 [17]

Type of data: Relational Databases.

Problem: User specifies a query as a set of keywords, and the system generates a set of candidate SQL queries to be executed on an RDBMS, in order to compute the answers.

Answer: The notion of an answer is a list of tuples having **all** the attributes of all the joined relations. (This can be interpreted as the JTTs of Spark, presented in Section 3.1.12.)

Ranking function: The results (tuples) are ranked by their likelihood given the SQL query that generated them. The SQL queries are in turn ranked by their likelihood of satisfying the user's keyword query. Given a tuple tree T resulting from the candidate network, CN^* involving the attributes, $A_1, A_2, \ldots A_m$, its score is calculated as follows:

$$P(T|CN^*) = \eta \times \frac{\sum_{i=1}^{m} \cos\left(\overrightarrow{A_i}, \overrightarrow{a_i}\right)}{|CN^*|}$$

where $\overrightarrow{A_i}$ represents the vector of all possible values for the attribute A_i in the database, $\overrightarrow{a_i}$ denotes the vector of all the values for attribute A_i that match the query, and η is a constant. The weights for the vectors are learned by adapting tf-idf to attribute values (but more complicated than the usual tf-idf computations).

3.1.16 • EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data, 2008 [18]

Type of data: Search in unstructured, semi-structured, or structured data modeled as graphs.

Problem: This paper claims to be the first to study keyword search for heterogeneous data sources, however, they don't make any attempt to interconnect different data sources in a graph.

Answer: These authors want to return **graphs**, not just trees. The motivation for this is that in some databases, e.g., biological ones, with protein-protein interactions, etc., users may be looking not just for a tree, but for a "neighborhood" (relatively small subgraph) in which the keywords appear. Such a neighborhood has the advantage of giving more information to the user.

To define this, they introduce a few notions:

- Given a graph G and a node $v \in G$, the **centric distance** of v, denoted $CD(v)$, is the maximal value among the distances between v and any node $u \in G$;
- The **radius** of a graph G, denoted $\mathcal{R}(G)$, is the smallest centric distance of every node in G;
- Given a query Q and a graph G of radius r:
  - a node $\omega \in G$ is called a **content node** if it contains one of the query keywords;
  - a node $\omega \in G$ is called a **Steiner node** if it is on a path $u \rightsquigarrow v$ such that u, v are content nodes;
  - the subgraph of G consisting of the Steiner nodes of G and the paths they are on is called a r**-radius Steiner graph**.

The r-radius Steiner Graph Problem is to find all the Steiner Graphs of radius at most r. Then, the top-k r-radius SGP is to find the k Steiner Graphs of radius at most r in the order of their relevance for the query.

Ranking function: There are some terms to consider when we are dealing with different sources of information. The first term considered is the compactness between two content nodes, computed as follow:

$$SIM(n_i, n_j) = \sum_{n_i \leftrightsquigarrow n_j} \frac{1}{(|n_i \leftrightsquigarrow n_j| + 1)^2}$$

where we sum for all path between a node n_i to n_j, and $|n_i \leftrightsquigarrow n_j|$ is the length of the path.

To measure the relevance between the input keywords (and not just the nodes) the next formula extends the previous one:

$$SIM(< k_i, k_j > | SG) = \frac{1}{|C_{k_i} \cup C_{k_j}|} \cdot \sum_{n_i \in C_{k_i}; n_j \in C_{k_j};} SIM(n_i, n_j)$$

where SG is the Steiner graph and C_{k_i} is the set of content nodes that contains k_i, also the $|C|$ means the number of nodes.

As most part of these therms are precomputed, for an evaluation for the compactness of an answer SG to query K:

$$Score_{DB}(K, SG) = \sum_{1 \leq i \leq j \leq m} SIM(\langle k_i, k_j \rangle | SG)$$

And the final model is:

$$Score(K, SG) = \sum_{1 \leq i \leq j \leq m} Score(\langle k_i, k_j \rangle | SG)$$

where

$$Score(\langle k_i, k_j \rangle | SG) = SIM(\langle k_i, k_j \rangle | SG) * (Score_{IR}(k_i, SG) + Score_{IR}(k_j, SG))$$

$Score_{IR}$ was presented before and is a TF-IDF based, with the formulas below:

$$ntf_{(k_i,G)} = 1 + ln(1 + tf_{(k_i,G)})$$
$$idf_{k_i} = ln(\frac{N+1}{N_{k_i}+1})$$
$$ndl_G = (1-s) + s * \frac{tl_G}{avg_{tl}}$$
$$Score_{IR}(k_i, SG) = \frac{ntf_{(k_i,G)} * idf_{k_i}}{ndl_G}$$

Where $tf_{k_i,G}$ is the term frequency in G. N is the number of r-maximal radius graph and N_{k_i} are the ones that contains k_i. tl_G is the total number of terms in G and avg_{tl} is the average number of terms among all such r-radius graphs.

3.1.17 • KEYWORD SEARCH ON EXTERNAL MEMORY DATA GRAPHS, 2008 [19]

This is a follow up on BANKS1 and BANKS2 to also consider external memory graphs. There is no change to the problem nor scoring function, instead, the algorithms deployed are different.

3.1.18 • STAR: STP APPROXIMATION IN RELATIONSHIP GRAPHS, 2009 [20]

Type of data: They consider **undirected graphs** with labels on nodes and edges and weights on edges. However, an extra, more or less hidden assumption is made: namely, that there exists a **taxonomy** in the graph, which enables finding a first connecting tree very fast. This first answer will then be improved upon (that is the nature of the approach).

Problem: They solve (approximatively) a Steiner Tree Problem, that is: given a set of k

nodes, find the smallest-cost tree connecting those nodes. This is different from the majority of works considered here, which address variants of the GSTP.

Answer: Given a **set of nodes** (not of keywords), they are interested in finding **the (single) tree connecting them** which has the minimum cost, where the cost is defined as the sum of the edge weights. This definition of cost is an important aspect of their approach (all their technique depends on it).

They also provide an extension to return top-k results (the best plus more trees), but they do not provide guarantees about the other trees.

Ranking function: They recall that the problem is NP-hard and therefore solve it with an approximation: they guarantee a solution whose cost is at most $O(\log(n))$ higher than the true cost of the (single, ideal) solution.

3.1.19 • Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data, 2009 [21]

Type of data: They consider **RDF (directed) graphs**. In particular, they identify **type** and **subclass** edges (not other special RDF properties).

Problem: A query is a set of keywords. They also allow the keywords to match **on the edges** (as opposed to numerous methods that only matched on the nodes).

Answer: An answer is a **SPARQL query** (not data). In particular, they consider conjunctive SPARQL queries.

Ranking function: They state that the score of a candidate query reflects the extent to which it matches the user's need. They introduce several score functions in the paper section *Scoring*, but unfortunately, they are tied to particularities of their algorithms and methods (they would not have been able to define the score earlier on). They use elements such as: the number of edges (in a graph, not in a tree, since their queries are computed from graphs); popularity of a node (resp., edge) label in the graph; the quality of the match between the node/edge label, and a keyword.

The authors' approach and intuition behind their cost metric is quoted here as we found it expresses things well: *"the costs of individual paths are computed independently, such that if the paths share the same element, the cost of this element will be counted multiple times. This has the advantage that the preferred graphs exhibit tighter connections between keyword elements. This is in line with the assumption that closely connected entities more likely match the users' information need."*

3.1.20 • Structured data retrieval using cover density ranking, 2010a [22]

Type of data: Relational databases

Problem: Keyword search.

Answer: The definition of an answer and evaluation of a query are the same as the technique described in Section 3.1.6 – the answers are JTTs.

Ranking function: They have adapted the Cover Density proposed over unstructured texts to structured data. The scheme proposed is generalized to work for any structured data. They introduce:

1. **Document:** A document, D, in structured data is defined as a concatenation of various fields, d_i. This means that a document is essentially a tuple from a candidate network, CN. However, the ordering of the fields matter in the final ranking (and therefore, the ordering of the fields after joins).

$$D = \{d_1, d_2, \ldots, d_{|D|}\}$$

2. **Structured Extent:** A structured extent, E, of a document D is a subset of the fields in D. That is, $E \subseteq D$. A structured extent satisfies a term set $Q' \subseteq Q$ if all the terms of Q' appear in E.

3. **Structured Cover:** A structured cover is a structured extent that satisfies Q' and does not contain a smaller subset of fields that also satisfies Q'. The set of all structured covers of a structured document D that satisfy the largest subset of Q is denoted by C_S.

4. **Scoring a document:** Each structured document, D, is scored as per its covers as follows:

$$score(C_S) = \sum_{E \in C_S} score(E)$$

$$score(E) = \begin{cases} \frac{H}{|E|}, & \text{if} |E| > H \\ 1, & \text{otherwise} \end{cases}$$

where $H \in [1, \infty)$ is a tuning parameter. Decreasing H rewards documents that contain smaller subsets of fields that completely satisfy a query's term set.

Essentially, the JTTs are scored in an IR-style by favoring tuples that contain maximum possible query terms and are shorter in length.

3.1.21 • Ranking support for keyword search on structured data using relevance models, 2011 [23]

Type of data: They work on directed graphs where they distinguish resource nodes (non-leaves) from attribute nodes.

Problem: Keyword query.

Answer: They say a keyword k matches a resource r if k matches at least one attribute of r or at least one edge going from r to one of its attributes. Then, an answer is a minimal rooted directed tree that contains at least one resource matching every query keyword.

Ranking function: This paper is mostly, first and foremost about the ranking. We cite below the authors' motivation as it is very clearly stated:

> *"The major shortcomings of previous work can be summarized as follows:*
>
> - *The **minimal distance heuristic** behind proximity search is rather convenient for the efficient computation of results but does not directly capture relevance.*
> - *The adoption of **IR-based ranking** is also problematic because unlike document ranking, the score of a result in this setting is an aggregation of several resources' scores. Combining resources with high "local scores" however, does not always guarantee highly relevant final results. Recent evaluation results [32] suggest that the proposed normalization methods are not effective in dealing with this issue.*
> - *More importantly, previous work implicitly assumes that **relevance is completely captured by the keyword query that is mostly short and ambiguous**. We consider this assumption to be too strong especially in this setting, where users might not be aware of the underlying structure and terminology of the database and thus, cannot specify complete queries."*

To address these shortcomings of previous work, the authors propose a **Relevance Model** based on the idea that for a given information need, queries and documents relevant to that need can be viewed as *random samples from the same underlying generative model*. Formally, an RM is defined as:

$$RM_R(v) = P(v|r_1, \dots r_m) = \frac{P(v, r_1, \dots, r_m)}{P(r_1, \dots, r_m)}$$

where R encapsulates the relevance.

Given a collection of documents C and the vocabulary of terms V, for a given vocabulary term $v \in V$ and document D, they define:

$$P(v|D) = \lambda_D \frac{n(v, D)}{|D|} + (1 - \lambda_D)P(v|C)$$

where $n(v, D)$ is the count of the word v in the document, $|D|$ is the document length, and $P(v|C)$ is the background probability of v (presumably this is the frequency of v in the whole corpus C).

Then, they call *pseudo-relevance feedback (PRF)* a **set F of documents derived from the query**, and use them as an **approximation of** R. Assuming the query terms are independent, the authors recall a notion of **relevance of a term for a query** introduced in prior work:

$$RM_F(v) \approx P(v|Q) = \sum_{D \in F} (P(D) \cdot P(v|D) \cdot \prod_{q_i \in Q} P(q_i|D))$$

Then, **the score of a document** $D \in C$ is based on its cross-entropy from the relevance model RM_F, defined as:

$$H(RM_F||D) = \sum_{v \in V} logP(v|D) \cdot RM_F(v)$$

3.1.22 • SPARK2: Top-k Keyword Query in Relational Databases, 2011 [24]

Type of data: Relational databases.

Problem: Their queries accept AND as well as OR semantics (AND here must have all keywords, and OR may have all keywords, default is OR).

Answer: The result of a keyword search query is a tree, T, of tuples, whose leaf nodes contain at least one keyword, this is also called a JTT (joined tuple tree). The size of a JTT is the number of nodes in the tree.

Ranking function: The same as in the previous SPARK work; the contributions here are more related to algorithms that solve the problem.

3.1.23 • Fast approximation of Steiner trees in large graphs, 2012 [25]

Type of data: They work on an **undirected, unweighted graph, assumed connected** (they say the approach can be extended easily to directed, weighted graphs).

Problem: They work on an **undirected, unweighted graph, assumed connected** (they say the approach can be extended easily to directed, weighted graphs). Given a set of **nodes**,

the ideal answer is the minimal-size Steiner tree (the one with the fewest edges) connecting the give nodes.

Answer: The ideal solution is the smallest Steiner tree (the one having the fewest edges). Since it is hard to find, they seek to find approximations by exploiting hubs in the graph, and indexes.

Ranking function: The (only) solution they find is a Steiner tree "as small as possible".

3.1.24 • Language models for keyword search over data graphs, 2012 [26]

Type of data: Graphs.

In the graphs, nodes and edges have three semantic fields: *title* field, which comprises name and types attributes; *content* field, which comprises all attributes and their values; and *structure* field, which comprises the value of the type attribute as well all other attributes without their values.

Problem: Keyword search on the *content* fields of nodes or edges.

Answer: Answers are nonredundant subtrees containing that include the given keywords.

Ranking function: Their ranking function is based on a **language models**, as follows. The probability that a query $Q = (q_1, ..., q_n)$ is generated by a field f of an answer A is:

$$P(Q|A_f) = \prod_i ((1 - \lambda) * P(q_i|A_f) + \lambda * P(q_i|Q))$$

Where the probabilities in the right side are calculated using the maximum-likelihood estimate as follow:

$$P(q_i|X) = \frac{tf(q_i, text(X))}{\sum_{t \in text(X)} tf(t, text(X))}$$

Here tf is the term frequency and $text(X)$ is the text related to X, this is all the text of nodes and edges from X concatenated.

Normalizations: For this part, the paper considers that ranking will use the probability of language models and the weights of semantics connections together, then these two measures have to be normalized to [0,1]. As lower semantic weights mean more correlation, this pattern will be used also for the probabilities that will be inverted and normalized as follow:

First probabilities in log form to avoid underflow:

$$R(Q, A_f) = \sum_i ln((1 - \lambda) * P(q_i|A_f) + \lambda * P(q_i|Q))$$

where f is a field such as content or title. And the inversion and normalization used is:

$$lscr^{ir}(Q, A_f) = 1 - \frac{1}{ln(-R(Q, A_f) + R_{max} + 2.781)}$$

where R_{max} is the maximum find in the top K solution founded and lscr is to emphasize that lower scores are better. For the final score for the language model, a linear combination of the content and title fields is used:

$$lscr^{ir}(Q, A) = \alpha * lscr^{ir}(Q, A_{title}) + (1 - \alpha) * lscr^{ir}(Q, A_{content})$$

Graph Weights: For the weights of the graphs, there are two criteria, semantic strength (static weights) and relevance to the given query (dynamic weights)

Static weights of nodes: For the static weights of nodes, they consider nodes with grater in-degree are more important and used the next formula:

$$w(v) = \frac{1}{ln(1.718 + idg(v))}$$

And for isolated nodes or nodes with no incoming edges, the value 1 is assigned.

Static weights of edges: For this one we consider the uniqueness of an edge, for describe this, they've used the concept of a similar edge as one edge with one of its endings be equal and the other ending of the same type. And the weight for an edge $e = (u, v)$ with $fdg(e)$ being similar to e and starting in u, and $tdg(e)$ being the similar edges of e ending in v, e is counted in fdg and tdf, is:

$$w(e) = 1 - \frac{1}{ln(0.718 + fdg(e) + tdg(e))}$$

Dynamic weight of edges To generate answers for a query $Q = (q_1, ..., q_n)$, a node is created for each q_i and connected to all nodes v that contains this word. If the word appears on the structure field of v, the weight of the edge e from q_i to v is 0, if q_i does not appear in the structure field the weight for the edge e is the same as the l-score of v with respect to the whole query Q as given by $lscr^{ir}(Q, A)$.

Scoring Answers

For the structural weight of a an answers, they define $W(Q, A)$:

$$W(Q, A) = \sum_{v \in V} w(v) + \sum_{e \in E} w(e)$$

Then this formula needs to be normalized before being added to the IR l-scores, with the following normalization:

$$lscr^{s}(Q, A) = 1 - \frac{1}{ln(W(Q, A) - W_{min} + 2.718)}$$

For the final score we combine the IR l-score with the structural l-score with a linear combination:

$$lscr(Q, A) = \beta * lscr^{r}(Q, A) + (1 - \beta) * lscr^{ir}(Q, A)$$

3.1.25 • Scalable Keyword Search on Large RDF Data, 2014 [27]

Type of data: They work on RDF graphs, which they assume to be very regular: *entity* nodes are typed by having attached *type* nodes and they also have *keyword* nodes.

Problem: They work on RDF graphs, which they assume to be very regular: *entity* nodes are typed by having attached *type* nodes and they also have *keyword* nodes. It is not clear if a keyword node can have keywords (if it is possible to have some attributes but lack a type – doesn't seem so).

Answer: Given a set of keywords, an answer is a root node plus a set of nodes such that each node matches one of the query keywords and the root is reachable from all the keywords (this is what they write, but later on they want the root to reach all the keywords; judging by the examples, they appear to consider the graph undirected).

Importantly, they also add that there only consider **one answer per root node** (other previous papers also did it the same assumption).

Ranking function: The score of a result is the number of edges it contains.

3.1.26 • Match-Based Candidate Network Generation for Keyword Queries over Relational Databases, 2018 [28]

Type of data: Relational databases.

Problem: Keyword search.

Answer: In keeping with the prior literature, an answer is a joining network of tuples.

Ranking function: They first manually generated *golden standard* answers for all queries of the datasets, based on the available query descriptions, then (post-hoc) computed result quality as the mean average precision with respect to the hand-made golden standard.

3.1.27 • Root Rank: A Relational Operator for KWS Result Ranking, 2019 [29]

Type of data: Relational databases.

Problem: Keyword search.

Answer: The notion of an *answer* in this paper is the same as that for Spark (Section 3.1.12).

Ranking function: The scoring scheme is the same as what has been proposed by the Labrador system (Section 3.1.15). Notably, they identify this scoring scheme to have Result Set Dependent (RSD) characteristic, that is, the distribution of keywords in the query results influences the ranking. The contribution of the paper is an implementation of the scoring scheme as an operator named Root-Rank, to be used after the join operator in a relational DB.

3.1.28 • Operator implementation of Result Set Dependent KWS scoring functions, 2020 [30]

Type of data: Relational databases.

Problem: This paper is an extension of previous work by the authors [29] (refer Section 3.1.27). The notion of an **Answer** and **Answer Score** remain the same. They implement and integrate one more operator named *Join-Rank*. The Join-Rank operator was introduced at the top of the query execution plan tree to perform concurrently the ranking process of the top-k results and the topmost join, this provides improved performance compared to the previous Root-Rank operator of their previous paper.

Answer: Same as the previous work by the authors [29] (refer Section 3.1.27)

Ranking function: Same as the previous work by the authors [29] (refer Section 3.1.27)

3.1.29 • Graph-based keyword search in heterogeneous data sources, 2020 [1]

Type of data: This work considers graphs that are built out of different kinds of data sources (relational, XML, JSON, RDF, etc.). The graphs are enriched through information extraction: when an entity is identified in a string, an entity node is created as a child of the string node, connected to it through an extraction edge. Edges have a *confidence* (for most, this is 1.0, but it can also be smaller). Further, nodes in the graph can be *equivalent*, while retaining different IDs (each ID reflects the data source where the node comes from and the position of the node).

Problem: Keyword search.

Answer: An answer is a minimal tree (set of edges connected together in a tree), matching all the query keywords. An edge can be taken in its original or in the opposite direction. The tree root is not significant. As usual, minimality means that removing an edge should make the tree miss one or more keywords. A further condition is that if a keyword is matched by two nodes in the tree, the nodes should be equivalent. It is possible for the same keyword to be matched by two nodes that are just *similar* but not equivalent.

Ranking function: They score is:

$$score(t, Q) = \alpha \cdot ms(t, Q) + \beta \cdot \prod_{e \in E} c(e) + (1 - \alpha - \beta) \cdot \prod_{e \in E} s(e)$$

where $0 \leq \alpha, \beta \leq 1$, $0 \leq \alpha + \beta \leq 1$ and:

- The *matching score* $ms(t)$ reflects how well its leaves match the query terms.
  - It is computed as the average of the match quality of t for each query keyword k, denoted $ms(t, k)$.
  - In turn, $ms(t, k)$ is the mean of the score of each tree *node or edge* wrt k.
  - The score of one node (or edge) wrt k is computed using a lexical distance between the node (edge) label and the keyword k. The distance function used is Levenstein if the mean length of (label+keyword) is at least 10, and Jaro otherwise.
- $c(e)$ is the confidence of an edge e;
- For a given node n and label l, let $N^l_{\rightarrow n}$ be the number of l-labeled edges entering n, and $N^l_{n\rightarrow}$ the number of l-labeled edges exiting n. The **specificity** $s(e)$ of an edge $e = n_1 \xrightarrow{l} n_2$ is defined as:

$$s(e) = 2/(N^l_{n_1\rightarrow} + N^l_{\rightarrow n_2}).$$

The weights α, β are provided by the user. By default, $\alpha = \beta = 0.2$.

3.1.30 • Efficient Computation of Semantically Cohesive Subgraphs for Keyword-Based Knowledge Graph Exploration [31], 2021

Type of data: They work on (directed) RDF graphs.

Problem: Keyword query.

Answer: An answer is a minimal tree whose leaves contain all the keywords. (They also say that this could be extended to have matches on edges by transforming the graph so that all the edges are unlabeled, and each original labeled edge becomes a new labeled node.)

Ranking function: They introduce:

- A **node weight function** `wt` which maps any node to a real, non-negative number
- A **semantic distance function** `sd` which maps any pair of nodes to a real, non-negative number. This distance can be orthogonal with respect to the graph structure, i.e., it can be based on their labels, the linguistic context surrounding them, or anything else.

- The total cost of an answer $T = \langle V_T, E_T \rangle$ as:

$$\texttt{cost}(T) = \alpha \sum_{v \in V_{>T}} \texttt{wt}(v) \; + \; (1-\alpha) \sum_{v_i, v_j \in V_T, i<j} \texttt{sd}(v_i, v_j)$$

With this, their problem statement is to find the minimum cost tree according to the above cost function. They call this the **quadratic Group Steiner Tree Problem (GSTP)** because there is the quadratic term in the cost formula. They say this problem is also NP-hard.

They provide two approximate algorithms for this: one with approximation ratio $O(m^2)$ (and computational cost in $O(m|V|^3|E| + m^3|V|^4)$), and another one with $(g-1)^2|V|$ approximation ratio - this is so bad it's almost comical - but which runs faster in practice (even though the worst-case complexity is the same).

3.2 Conclusion

In the above, we have shown that the literature comprises a large number of proposals for keyword search on graphs. The *graphs* are either native (RDF graphs, property graphs, Entity-Relationship graphs, weighted graphs, etc.), or are graph views of other kinds of data: the most popular models are relational and XML, but JSON, CSV, and other formats have also been considered.

A *query* in these proposals is a set of keywords, with (in a majority of works) AND semantics (all keywords must be matched in a result). A few proposals enlarge this perspective to also consider OR semantics (some keywords are optional).

For what concerns the *answers*, in a wide majority of proposals from the literature, answers are trees consisting of nodes and edges from a graph; on the contrary, a few works return subgraphs that are not trees, based on the idea that a graph provides more information on how nodes are related.

Finally, each *scoring function* is an attempt to map goodness (from the perspective of answering an information need) with criteria that can be measured/computed on each answer. As we have shown, the main ingredients for scoring functions are: matching quality, IR-style components such as TF/IDF, graph scores such as PageRank, more advanced linguistic models, and metrics, etc.

There has been no comprehensive comparison of the *quality (appropriateness) of these scoring functions* so far. Several reasons for this can be identified:

- It is inherently difficult to estimate IR result quality: this requires human judgment.
- The database community (which has produced most of the above work) for such quality-oriented studies does not have a strong culture in quality evaluation. Indeed, an overwhelming majority of the experimental validations performed in the above works focus only on the speed at which answers are found.
- Different authors may have had in mind different notions of what a "good result" is, making comparisons inherently difficult.

- The differences in the nature of results targeted by different works (trees, graphs, SQL queries, SPARQL queries...) are also complicating the comparison task.
- Last but not least, the "artefacts" of each kind of data graphs (e.g., RDF graphs look quite different from relational databases viewed as graphs) also make the comparison challenging.

In this context, our literature survey above aimed to gather the various methods and ideas in a single place, to facilitate the discovery of the literature as well as future work in the area.

4 EFFICIENCY ON BENCHMARKS

In this section, we report on efforts we carried to evaluate the effectiveness of ConnectionLens on an existing benchmark for keyword search [32]. While all the works surveyed in the previous section had an experimental evaluation, they used various datasets and evaluation metrics, therefore their results are not really comparable.

The benchmark introduced in [32] is defined over relational databases, thus it can be seen as a particular case from the perspective of ConnectionLens. However, because it is the first (and, to date, the only) systematic comparison, and because ConnectionLens does aim at covering also relational databases, we found it useful to see how ConnectionLens measures on it.

Below, we start by reviewing the content in the paper [32]. Then, we describe: how the methodology they created was very insightful; the difficulties we encountered running the benchmark queries in ConnectionLens; finally, how ConnectionLens performs compared to the other systems presented in the paper.

4.1 A framework for evaluating database keyword search strategies, 2010b [32]

This paper proposes a benchmark dataset for evaluating keyword search algorithms. It also performs a comparative evaluation of various previously proposed keyword search algorithms, specifically those covered in Sections 3.1.1, 3.1.2, 3.1.6, 3.1.8, 3.1.10, 3.1.12, 3.1.13, 3.1.14, 3.1.20. They categorize existing systems into two types, depending on their ranking approach: *proximity-based* search (and ranking), vs. *Information Retrieval* style ranking.

To evaluate a keyword search algorithm, the authors consider the following four metrics:

1. The number of **top-1 relevant results** is the number of queries for which the first result is relevant. Here, a set of relevant results is for each benchmark query is manually established by the authors and comes as part of the benchmark.

2. **Reciprocal rank** is the reciprocal of the highest-ranked relevant result for a given query, which is the inverse of the numerical position, i.e. if the first relevant answer is in the first place the reciprocal rank is 1.0, if it's in the second place 0.5, and so on.

3. The **average precision** for a query is the average of the precision values calculated after each relevant result is retrieved (and assigning a precision of 0.0 to any relevant results not retrieved).

4. **Mean average precision (MAP)** averages the average precision values across information needs, to derive a single measure of quality across different recall levels and information needs.

Additionally, they also measure the **correlation between the results returned by the various systems** by computing the normalized Kendall distance. The Kendall distance between two permutations is the number of pairwise swaps needed to convert one permutation into the other.

Their conclusions are:

1. IR-style ranking schemes prefer larger results that contain additional instances of the search terms, to smaller results matching the query. Hence, they fail for queries with exactly *one* relevant answer.

2. Scalability remains a significant concern for the proximity search systems. Most such systems are unable to converge on large datasets.

3. IR-style scoring functions (particularly the scheme described in Section 3.1.6) outperform the proximity search systems on graphs with large text labels, because their scoring functions were designed for lengthy unstructured text. The only exception is the BANKS system (Section 3.1.1), whose ranking, based on node prestige and edge weights, roughly captures the best of both worlds.

4. The results from compared systems are only moderately correlated at best.

We looked for the benchmark itself but the link in the paper is broken as the Ph.D. author has finished his Ph.D. and left his lab. Then we found `https://joel-coffman.github.io/resources.html`.

4.2 Insightful ideas

Some insightful ideas about this paper could help the development in the field and they were very useful in this work.

- The authors noted divergences in existing experimental evaluations across the published papers;

- They proposed a methodology to evaluate retrieval effectiveness and repeating evaluations from existing papers;

- Many existing techniques showed did not scale on moderately size datasets.

- The authors note that TREC, the Text Retrieval Conference, significantly improved their field through proposing a standardized evaluation, thus their proposed benchmark has similar ambitions for their respective field.

4.3 Benchmark methodology

In accordance with the IR literature [35, 36], the authors of [32] built a set of **50 information needs**, 50 being considered in the IR field a traditional minimum for a comprehensive evaluation of a search system. Following [35], they argue that using a representative of real-world queries is more interesting than drawing them at random because randomly chosen queries usually do not reflect the distribution of real users' information needs. To this goal, the authors have first picked a set of topics that could interest users of the system holding a certain database, then try different queries related to the same topic.

They repeat the process of evaluation on 50 different queries in three different datasets, to avoid results that are too dependent on a particular dataset. They apply a binary relevance assessment, where each returned result is classified as either *relevant* or *nonrelevant*. The authors prefer this simple evaluation to a more fine-grained one where some results may be considered more interesting than others, in order to avoid the subjectivity inherently present in such interestingness assessments.

The three datasets they introduced are:

- Mondial: a collection of geographical and demographic information.
- IMDb: information about movies, their actors, directors, etc. As this dataset is rather large, the authors have taken some steps to shorten it, as they describe in the paper.
- Wikipedia: this is a set of 5500 Wikipedia articles, structured in a relational database reflecting the articles, the users, and links between pages.

The metrics explained in Section 4.1 are computed for 50 queries on each dataset, for the top 1000 results of each system.

4.4 Difficulties encountered

4.4.1 • Data loading

The author's Web site `https://joel-coffman.github.io/resources.html` provides relational database dumps, together with answers for each query, and topics together with all their expected relevant answers. The database dumps separated the schema from the data file. We were initially not able to load the data, after having loaded the schema successfully. After investigation, we find that the schema included primary key - foreign key constraints that were creating cyclic dependencies between the different tables to load. The final solution to import data was to first import the data without any of the primary key-foreign key constraints, thus eliminating the cyclic dependency, and then adding the constraints back to the resulting database.

Once these issues were solved, trying the benchmark on ConnectionLens has exposed several problems:

Listing 1: Answers proposed by the benchmark paper to the queries "Thailand" and "Mongolia China" on the Mondial dataset

```
# thailand
([3494], [])

# mongolia china
([3500, 3477, 110], [(110, 3500), (110, 3477)]) # borders
```

1. The system scaled poorly when loading relational data. This is because that part of the code, developed in 2018, had since not been improved; it has been tested for correctness, but not for performance. Indeed, recent datasets built in ConnectionLens [37, 38] stress-tested the JSON, HTML, XML, and RDF ingestion, but not the relational one.

2. The system was unable to handle cases where a primary key - foreign key consisted of more than one attribute. Conceptually, from the graph search perspective, this should not make any difference, but concretely, the ConnectionLens code failed to accommodate some of the benchmark data where several attributes formed a key *together*.

These issues were exposed due to my internship, and they have consequently have been fixed by the team, improving the overall quality of the software. However, they have slowed down my work to some extent.

4.4.2 • Comparing answers

Since the benchmark has been designed for relational data, an answer in their framework is not a tree, but rather a Joined Tuple Tree (JTT). Within ConnectionLens, each tuple leads to a graph node, and there are edges between the nodes connected by primary key - foreign key nodes. However, some post-processing is needed to go from a ConnectionLens *tuple-derived node* to the *relational database tuple* that is part of the benchmark's expected solution. Note, for instance, that some attributes of a tuple part of a benchmark JTT may not be present at all in the CL answers, that have *finer granularity* (in particular, an answer may include a tuple node and a node issued from one of its attributes, but not the whole tuple).

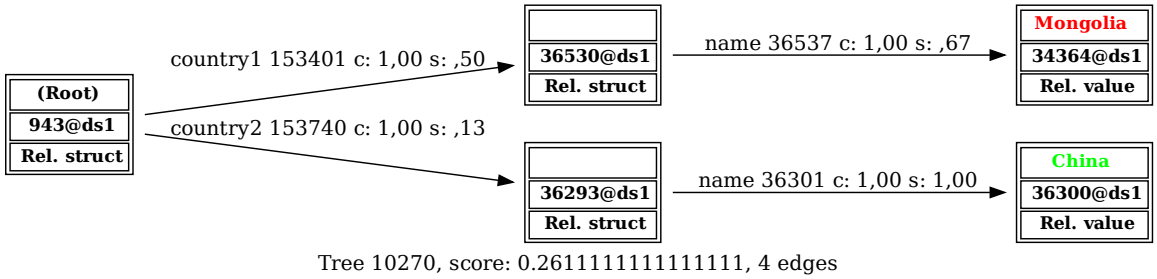
To overcome this difference, I inspected all answers myself and considered some adaptations as valid for each answer. This is illustrated by the examples below.

Listing 1 shows two sample queries, proposed and the benchmark's gold standard answers for them In parenthesis there are two arrays corresponding the first to nodes, and the second to edges. The number inside the brackets for nodes are *row counts saying where a rows come from (in the dataset)*; the pair of indexes inside the brackets of edges correspond to the nodes connecting the edges.

The first example in the listing 1 is the query "Thailand". The referenced answer here is just a tuple node. In Figure 5 there is also just one node, thus, clearly, the answers are equivalent. The difficulty in automatizing this check is to retrieve the index of the tuple in the original

Tree 0, score: 1.0, 0 edges

Figure 5: Graphical answer to the query "Thailand" in ConnectionLens from Mondial dataset



Tree 10270, score: 0.2611111111111111, 4 edges

Figure 6: Graphical answer to the query "Mongolia China" in ConnectionLens on the Mondial dataset. The topic of this query is the borders between countries.

dataset. When data is imported to ConnectionLens, all the information in a row is separated into different nodes, making it difficult to compare answers[3].

The second query "Mongolia China" illustrates also a second issue we encountered. The figreffig:mongoliachina shows the equivalent answers I proposed to be the same as the one proposed by the author, since the author just uses tree nodes, two for country rows and one for the border that links those previous countries, and the two edges between the three rows. In ConnectionLens, the solutions use 5 nodes and 4 edges, since the connection between information is made undirectly, through a foreign key. In this case, the node of the border is connected through two nodes (labeled country1 and country2) to the nodes that matched "Mongolia" and "China".

To run the benchmark on ConnectionLens, I decided to accept the answers in Figures 5 and

[3]It is slightly ironic that the authors of [32] also wrote that they have not tried to re-implement some systems to be compared, because that seemed too difficult.

6 as correct, since they reflect the best possible correspondence that ConnectionLens can find for the benchmark's gold standard answer. Finally, I have manually checked all answers.

4.4.3 • Issues uncovered in ConnectionLens' query answering

As part of the effort involved in meaningfully running the benchmark, I studied every query of every dataset to check if they worked as predicted or not. This lead to identifying a set of issues to find the answers for each query and improve the system to address these issues. We explain these below.

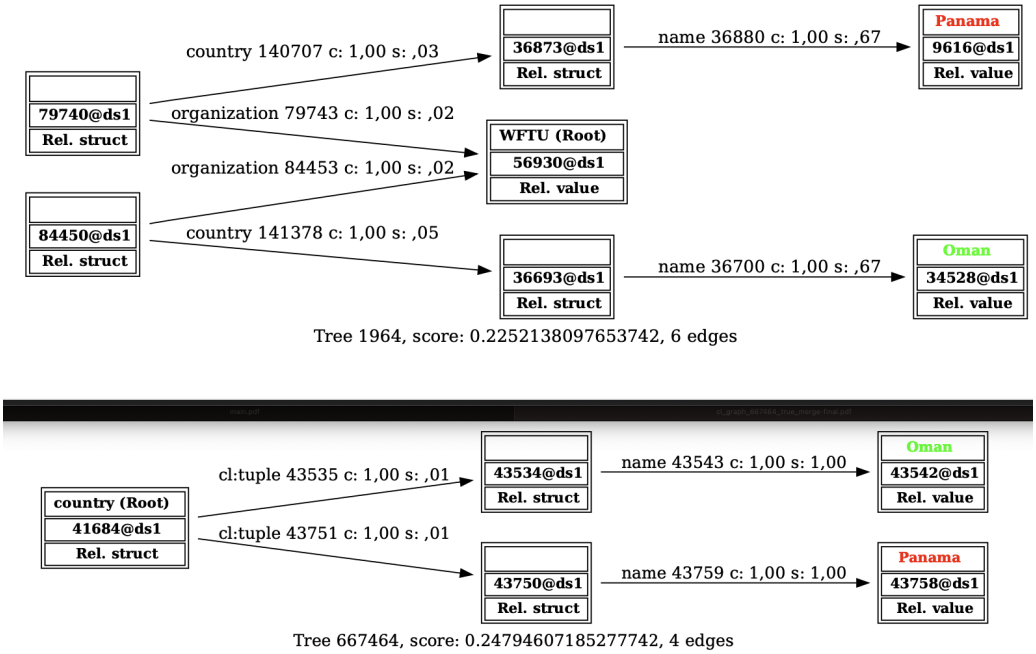


Figure 7: Two sample answers on the Mondial dataset with the query "panama oman".

- The topic "Countries belonging to the same organization" on the Mondial dataset has led to some queries such as "panama oman" and "poland cape verde organization". On such queries, ConnectionLens does not return the best answer first. For instance, Figure 7 shows two answers and their respective scores. Intuitively, the top answer is the one we prefer, since it uncovers an interesting connection between tuples, specifically, going through the country table. However, ConnectionLens does not rank this answer first; it gives it a slightly lower score than the answer shown underneath. We identified two types of issues.
  - It happens that the edge specificities going through this WFTU (The World Federation of Trade Unions) have *also have a low specificity*. Indeed, WFTU has 112 members, thus, 112 edges enter the node labeled "WFTU". Further, it turns out that there are 59 foreign keys labeled "country" pointing to Panama, in this dataset; this

also leads to a rather low specificity of those edges. The top (intuitively, better) answer chains two organization edges with two country edges, therefore, the specificity component in the score of this answer is quite low. This highlights that specificity alone does not suffice to identify the most interesting answer in this case.

  - The second issue is that the terms used to match a country name may match other database nodes, e.g., provinces, and answers that go through the province node may get a higher score due to lower specificity of the edges.

- Artist names in IMDb: Most artist names consist of two terms, e.g., "Julia Roberts". Here the issue when we are looking for the top-1 metric is that the node with the solution is labeled as "Roberts, Julia", but there are many other nodes with similar text, e.g., quotes that refer to the name of this artist, or a similarly named artist, e.g., we encountered "Roberts, Julia M.", which is distinct from the desired node. This is an issue caused by the similarity the algorithm calculates to each label to the query keywords, which makes the algorithm accept "Roberts, Julia M." as being as good (or better than) "Roberts, Julia". There are two possible solutions: the first would be to refine the similarity score; the second one would be to introduce a notion of *node prestige* (or ranking) in the score function.

- Stop words in the query: This is a problem that occurred a lot for quotes or movie names in IMDb, for example, the "lord of the rings". In ConnectionLens, stopwords such as "the" are not indexed, and as a consequence, ConnectionLens finds no answers for queries containing them. Several solutions to this could be envisioned.

  - The ConnectionLens query module should be modified to internally replace such a query with "lord rings", i.e., remove the stop words. This has not been done until the end of my internship.
    This is the best adaptation that could be envisioned between the benchmark's setting, and ours. However, this would turn a *sequence* of words into a *smaller set* of words and thus, presumably, alter some of the benchmark authors' intent.
  - Alternatively, ConnectionLens provides an *exact search* facility where one could ask to strictly match only nodes labeled "Lord of the Rings". We tried this for a few examples and could see that this enabled ConnectionLens to find some results. Yet, this also modifies the intent of the benchmark and only works when 1 graph node is labeled with the exact query terms (not in the more general case).

4.5 Benchmark result and evaluation

Recall that the score function used by ConnectionLens for scoring the answers trees was explained in 2.4.2. The benchmark metrics which allow measuring the quality of a given system were described in Section 4.1. Since the verification was made manually, because of the inconsistencies explained in Section 4.4.2, the mean average precision (mAP) and average precision were not calculated, since this was too tedious for 550 queries.

Mondial	Topics 1-50	
	Top-1	R-rank
ConnectionLens [37]	39	0.780
BANKS [3]	16	0.358
DISCOVER [4]	31	0.671
Efficient [13]	21	0.514
Bidirectional [10]	34	0.730
Effective [12]	22	0.495
DPBF [16]	37	0.823
BLINKS [15]	36	0.770
SPARK [14]	27	0.607
CD [22]	36	0.804

Table 1: Mondial results for top-1 score and the reciprocal ranking. All but ConnectionLens results are from [32].

Datasets	Mondial	IMDb	Wikipedia
	Topics 1-20	Topics 1-20	Topics 1-15
R-rank	1.0	0.5	0.503

Table 2: Reciprocal rank attained by ConnectionLens on each dataset of the benchmark.

4.5.1 • Results

The tareftab:mondialresults shows results of ConnectionLens on the 50 benchmark information needs of the benchmark, compared to the systems studied in [32], on the Mondial database. These are very good results: they show that ConnectionsLens is the best in terms of Top-1 results, and the second in terms of reciprocal ranking. This shows that in terms of the score function, with some modest additions, ConnectionLens could perform very well.

Figure 8 recalls the performance of the systems under test in [32]. For what concerns ConnectionLens, Table 2, shows that: ConnectionLens was the best in terms of reciprocal ranking for Mondial database for the 20 first queries compared to other systems; it also has reasonably good efficiency on IMDB where it was the third better system (but very far from the two best systems); finally, on the Wikipedia dataset, the result was quite poor (ranked 6 out of 10).

On IMDB, the lag of ConnectionLens is mostly caused by the stop word issue discussed above; these appear in 7 queries of the 20 studied in the table 2, and also there are 14 among the 50 queries. Probably, solving this issue would lead to better results.

On the Wikipedia dataset, the reciprocal performance could be improved as well. Most answers were founded, however, as the Wikipedia dataset has the description of the pages, the system identifies the text of a wikipedia page more similar to the query than the node which is the title of the page. For example, for the query "International English", the targeted keyword is written as "international_english" and received a score of 0.895, which is a (high) score given by label similarity computation. However, 2525 nodes received a (higher) score of 1.0, because

their label contained "international" and "english".

- On one hand, the similarity computation could be improved, not to penalize the underscore connecting words.
- On the other hand, this would only raise the title's rank to 1.0 but is insufficient to make it stand out among with respect to the 2525 other nodes achieving the same, high score. To go beyond, one would need to incorporate for instance some centrality or PageRank metric, in order to help the title stand out; or, modify the matching score component in order to penalize long labels, of which the query keywords are just a tiny part.

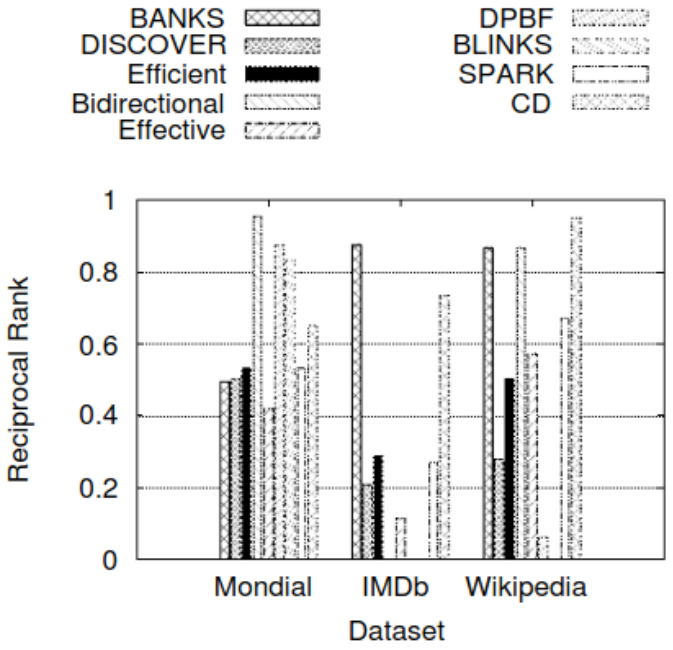


Figure 8: Reciprocal ranking on the Mondial, IMDb, and Wikipedia datasets for the first 20, 20, and 15 queries. This image is from [32].

4.5.2 • Conclusion

ConnectionLens has a great performance considering that the score function was proposed for structured, semistructured, and unstructured data, it's very impressive that the score function works as well as other score functions specially designed for structured data.

ConnectionLens worked well, but it still has a very simple score function and has space to test other solutions or to bring news ideas from other papers, such as the node prestige, or add proximity search measures to their score function.

A continuation of this work would be to test measures or to introduce techniques in the score function proposed in 3 and verify if their implementations could improve the results for

these datasets. Another improvement to test results, would be to find a way to transform the answers of the benchmark in answers in ConnectionLens since it was very difficult to do the verifications manually and that would make it possible to calculate the mAP for the datasets, that's another measure and is less noise than top-1 and reciprocal rank.

5
OPTIMIZATIONS IN SEARCH ALGORITHMS

Experiments with ConnectionLens (and in particular, some of the queries of the benchmark described in the previous Section) highlighted a particular kind of problematic queries. These occur when one or a few of the query keywords are extremely popular, e.g., "France" in a french-language corpus. GAMSearch, like any similar algorithm, grows exhaustively trees starting from the nodes that match each keyword. When one or a few query keywords have a huge number of matches, this entails that a lot of effort will be spent searching trees from these matches.

1. If *every* query keyword matches a very large number of answers, we are simply confronted with a huge search space, and there is not much we can do to help.

2. In contrast, if *some* query keywords match very few nodes, one could *privilege search starting from these rare keywords*, that is: invest more effort exploring the neighborhood of these "rare" matches, in the hope that we will encounter matches for the other query keywords. An sample rare keyword could be "intergouvernementalisations", which is the longest known French word[4]. *If* answers of moderate size exist, starting from this rare word is likely to encounter a node matching France. In contrast, not all the (many) nodes matching France are likely to have this word in their neighborhood, thus, search effort would be wasted if we exhaustively explore those.
   If, on the other hand, no such answers exist, we are no worse off than if we explored from all the matches with equal probability.

In this section, we describe a heuristic we proposed, aimed at improving performance in case 2. above.

As described in the section 2.4.1, the GAMSearch algorithm uses a priority Queue PQ as an auxiliary data structure to find the answers trees. PQ is filled with (tree, edge) pairs, such that the edge can be used to Grow a larger tree. To implement our heuristic, for a query of m keywords, assuming each keyword w_i is matched by N_i graph nodes, we create $2^m - 2$ priority queues, instead of a single one, one priority queue for each non-empty subset of the keywords (other than the complete query). For instance, if the query consists of keywords a and b, we create one priority queue for $\{a\}$ and the other for $\{b\}$; for a query with three keywords, we would create priority queues for $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$, respectively. Based on these, when the algorithm runs, instead of applying Grow on the pair at the top of the single queue PQ, we first *select the queue with the smallest size (fewest entries)*. As long as one or a few of the keywords appear in fewer trees compared with more popular keywords, they will be prioritized.

This idea has been implemented as a new algorithm, called OptiGAM, as a branch in the ConnectionLens project.

[4]Source: Wikipedia, see `shorturl.at/gsyQU`

5.1 Experimental validation

This optimization has been implemented towards the end of my internship and we were able to validate it only towards the end of the internship period. For the purpose of this validation, we decided to generate some synthetic graphs, controlling the following parameters:

- The total size of the graph;
- As a simplification, we focused on queries with just two keywords (a generalization would be easy). We want to control the number of nodes matching each keyword. We assume that the graph matches much more keywords than just the query keywords.

Further, we wanted a graph shape that makes it possible to reason at least approximately on the sizes of the search space, in order to understand the algorithm behavior.

For this purpose, we proceed as follows.

1. We generate XML documents with a controlled and rather simple structure.
   (a) Each tree root has N children, each of which is a full binary tree of depth d (counting node levels). Thus, there are $N \times 2^{(d-1)}$ leaf nodes.
   (b) Among the leaves, N_a are labeled *alpha*, and N_b are labeled *beta*; all the other leaves are labeled with random strings of length 3.
2. We ingest such a document in ConnectionLens, leading to a single graph.

The total number of nodes in such an XML tree is very easy to compute from N and d. Further, the query $\{alpha, beta\}$ has exactly $N_a \times N_b$ results; each result is of size $2 \times (d-1)+2$ since it needs to go from a leaf labeled *alpha* to the root, and from there down again to a leaf labeled *beta*. Every intermediary tree developed by GAM (or OptiGAM) is a path from one of the leaves labeled *alpha* or *beta*, toward another node of the tree. If the algorithm runs to completion, all such paths will be developed.

Figure 9 shows a sample XML document generated in this way with $N = 6$, $N_a = 1$, $N_b = 3$, and $d = 3$.

5.2 Experimental results: GAM vs. OptiGAM

In our experiments aiming to compare GAM and OptiGAM performance, we have used synthetic graphs generated from XML documents where we set $N = 100.000$ and $d = 2$. This leads to graphs with 500.001 nodes (the root, 100.000 children of the root, 200.000 children of these, and 200.000 value leaves). Each solution involves **6 edges**. These size parameters were chosen based on our experience with ConnectionLens, to make things challenging for the search algorithm. Further, to study the impact of the difference of frequency among the different keywords, we varied N_a and N_b; we have considered query answering until the *first*, respectively,

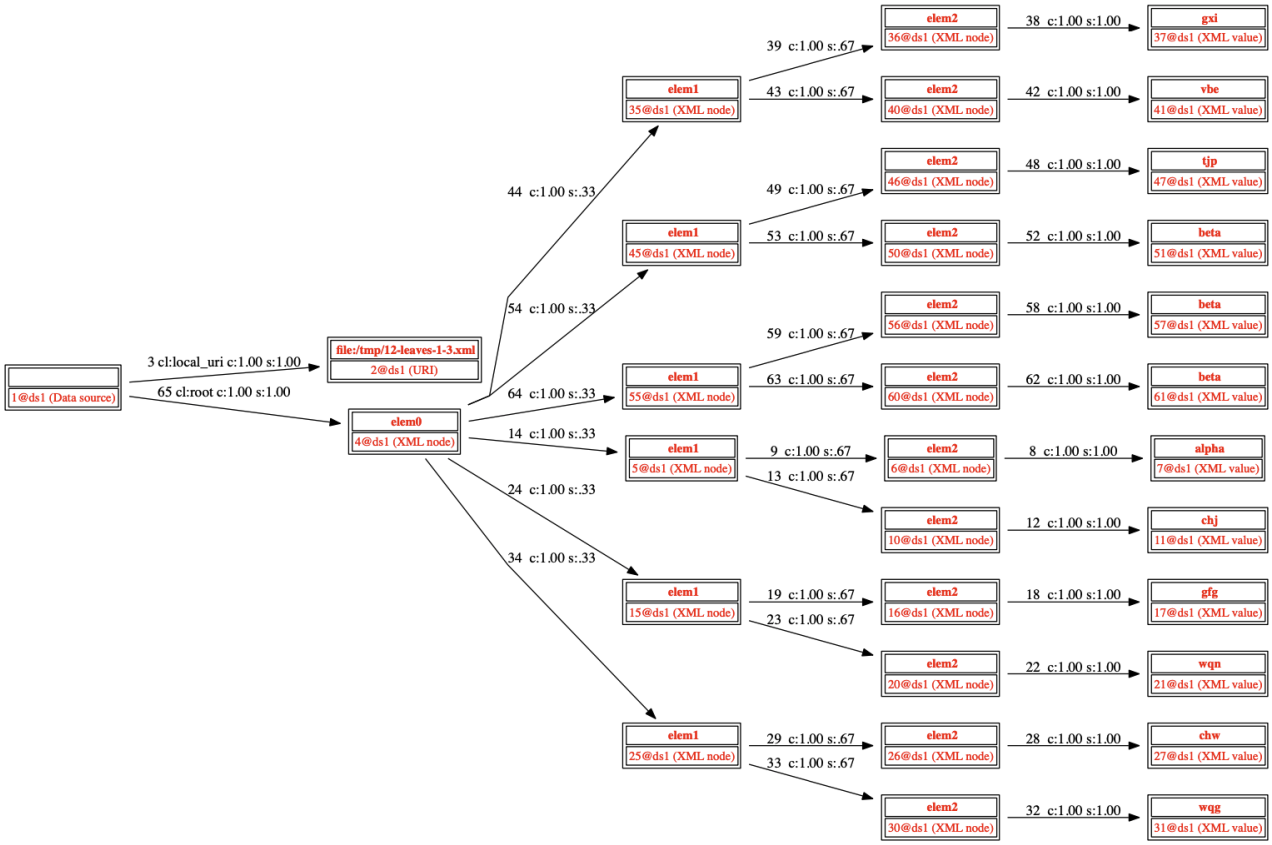


Figure 9: Sample synthetic XML document.

the *first 100* answers are obtained. In all cases, both algorithms ran with a timeout of 100 seconds.

Search space On this simple type of documents, we can enumerate (and measure) the search space, that is, the trees that GAM or OptiGAM need to build before finding all solutions, as follows:

- S_0 is a set of $N_a + N_b$ trees of 1 node
- S_1 is a set of $N_a + N_b$ trees of 2 nodes (each matching leaf with its parent element)
- S_2 is a set of $N_a + N_b$ trees of 3 nodes (each matching leaf with its parent and grandparent elements; each grandparent is a child of the root)
- S_3 is a set of $2 \times (N_a + N_b)$ trees (each tree from S_2, together with the other child of its root, and then with this child and its leaf node)
- S_4 contains: for each S_2 tree, a one edge larger tree that goes to the document's root element; further, each such tree can grow by one, then another edge toward the actual document root and the node indicating the original document URI. Thus, the size of S_4 $3 \times (N_a + N_b)$.
- S_5 contains: for each S_4 tree rooted in the root element labeled elem0:
  - $(N_a + N_b) \times (N - 1)$ trees going down one edge, towards a different sibling, an element labeled elem1;
  - based on the above, $2 \times (N_a + N_b) \times (N - 1)$ trees going one further edge down to an element labeled elem2;
  - finally, $2 \times (N_a + N_b) \times (N - 1)$ trees going one further edge down to a leaf. $N_a \times N_b$ among these are solutions.

Overall, the size of the search space which contains the sets S_0 to S_5 described above is:

$$(N_a + N_b) \times (1 + 1 + 1 + 2 + 3) + (N_a + N_b) \times (N - 1) \times (1 + 2 + 2) =$$

$$(N_a + N_b)(8 + 5 \times (N - 1)) = (N_a + N_b) \times (5 \times N + 3)$$

which is of the order of $5 \times (N_a + N_b) \times N$.

Hardware configuration The experiments were run in a machine with a processor Intel® Core™ i5-8265U CPU @ 1.60GHz × 8, a memory RAM of 16 GB, and the operating system Linux Ubuntu 20.04.

Table 3 shows the results for $k = 100$. For $N_a = 1$, GAM struggles and doesn't find any

Searching for the first 100 answers (timeout: 100.000 ms)						
			GAM		OptiGAM	
N_a	N_b	Search Space	Time (ms)	results	Time (ms)	results
1	100	50.500.000	Timeout	0	Timeout	94
	10.000	5.000.500.000	Timeout	0	50.772	100
	100.000	50.000.500.000	Timeout	0	Timeout	0
1.000	1.000	1.000.000.000	Timeout	6	Timeout	9
	2.000	1.500.000.000	Timeout	4	Timeout	9
	10.000	5.500.000.000	Timeout	0	Timeout	9
	100.000	50.500.000.000	Timeout	0	Timeout	0

Table 3: Search until finding the first 100 answers *or* reaching a timeout of 100 seconds (whichever happened first), for different synthetic documents, using GAM and OptiGAM. In gray the executions that were unsuccessful (no result returned before the code stopped).

answers before the timeout. For $N_a = 1.000$, with a lower (or no) imbalance between keyword frequencies in the graph, GAM works better (at least some solutions are found), and OptiGAM is no worse than it.

In comparison, OptiGAM significantly outperforms GAM for $N_a = 1$, retrieving most of the answers before the timeout for $N_b = 100$ and 100 answers for $N_b = 1.000$. For $N_a = 1.000$, OptiGAM still outperforms GAM, finding more answers before the timeout, but the number of results is very low in comparison to all the possible answers, this will be discussed next.

For $N_a = 1$ and $N_b = 100.000$, and when $N_a = 1.000$, both GAM and OptiGAM perform poorly, because the size of the search space increases. Still, we see OptiGAM is no worse than GAM.

Table 4 uses the same graphs as Table 3 but focusing on the time to the first answer.

For $N_a = 1$, OptiGAM confirms its advantage. An interesting observation is that this holds even for $N_a = N_b = 1$ where in principle we expect no difference. A possible explanation could be that having 2 smaller priority queues is more efficient than using a single one, given the large size it reaches. For $N_a = 1.000$, OptiGAM is also quite faster; this clearly validates its interest.

In conclusion, analyzing the Tables 3 and 4 one can conclude that OptiGAM had proven to be better than the actual GAM search in ConnectionLens, since it had a better or equal performance for all cases experimented for this report. These answers were not very easy to find (size 6). However, very large search spaces remain challenging for exhaustive exploration.

Searching for the first answer (timeout: 100.000 ms)						
			GAM		OptiGAM	
N_a	N_b	Search Space	Time (ms)	results	Time (ms)	results
1	1	1.000.000	46.502	1	34.359	1
	100	50.500.000	Timeout	0	35.350	1
	10.000	5.000.500.000	Timeout	0	50.379	1
	100.000	50.000.500.000	Timeout	0	Timeout	0
1.000	1.000	1.000.000.000	51.440	1	38.790	1
	2.000	1.500.000.000	97.267	1	40.957	1
	10.000	5.500.000.000	Timeout	0	51.901	1
	100.000	50.500.000.000	Timeout	0	Timeout	0

Table 4: Search until finding the first answer *or* reaching a timeout of 100 seconds (whichever happened first), for different synthetic documents, using GAM and OptiGAM.

6 CONCLUSION

The global focus of my 4 months internship has been query answering in ConnectionLens. First, the answer ranking aspect that has received less attention so far, and it was my task to see how good it is and how we could improve. Second, we aimed at making search more efficient for a family of cases, while not making things worse for the others.

For what concerns the ranking, given that ConnectionLens addresses (and generalizes) problems previously addressed within separate domains (just relational data, or just XML, or just RDF, etc.), I had to first gather all the interesting ideas proposed in the (segmented) existing literature, the efficiency of ConnectionLens, and optimization of the search algorithms. Each of these topics has its particularities and could be improved in future work.

The literature survey can be very helpful for researchers in the field, as it summarizes several papers in one place and describes parts of interest for each of them about the score function. As future work on this, this effort could be turned into a published survey.

The second part of my work on result ranking consisted of applying ConnectionLens to the most systematic benchmark built so far, which is based strictly on relational databases. This has allowed to identify and solve several issues related to this less-used part of the platform. This work has also allowed to see that ConnectionLens returns good (or the best) answers in some cases, but also to highlight some of its limitations. In particular, the specificity function whose goal was to help return non-trivial results does not suffice to always attain this goal. As discussed in the section 4.5.2, in the future, incorporating some notion of node rank based on the graph structure and/or on some text frequency statistics could improve its performance.

For the optimization of the search algorithm, the results were very encouraging to use the adaptation I've created to some particular cases I have studied. While the evaluation presented here is relatively short, it is quite encouraging; OptiGAM appears a very useful adaptation of GAM. More work is needed to confirm these results on more diverse, real-life datasets.

REFERENCES

[1] Angelos Christos Anadiotis, Mhd Yamen Haddad et Ioana Manolescu : Graph-based keyword search in heterogeneous data sources. *In BDA 2020 - 36ème Conférence sur la Gestion de Données – Principes, Technologies et Applications*, Online, France, octobre 2020. Informal publication only.

[2] Camille Chanial, Rédouane Dziri, Helena Galhardas, Julien Leblay, Minh-Huong Le Nguyen et Ioana Manolescu : Connectionlens: Finding connections across heterogeneous data sources. *Proc. VLDB Endow.*, 11(12):2030–2033, août 2018.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti et S. Sudarshan : Keyword searching and browsing in databases using BANKS. *In Proceedings 18th International Conference on Data Engineering*, pages 431–440, 2002.

[4] Vagelis Hristidis et Yannis Papakonstantinou : DISCOVER: keyword search in relational databases. *In VLDB*, 2002.

[5] Sanjay Agrawal, Surajit Chaudhuri et Gautam Das : Dbxplorer: A system for keyword-based search over relational databases. *In* Rakesh Agrawal et Klaus R. Dittrich, éditeurs : *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pages 5–16. IEEE Computer Society, 2002.

[6] Lin Guo, Feng Shao, Chavdar Botev et Jayavel Shanmugasundaram : XRANK: ranked keyword search over XML documents. *In* Alon Y. Halevy, Zachary G. Ives et AnHai Doan, éditeurs : *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 16–27. ACM, 2003.

[7] Vagelis Hristidis, Yannis Papakonstantinou et Andrey Balmin : Keyword proximity search on XML graphs. *In* Umeshwar Dayal, Krithi Ramamritham et T. M. Vijayaraman, éditeurs : *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 367–378. IEEE Computer Society, 2003.

[8] Vagelis Hristidis, Luis Gravano et Yannis Papakonstantinou : Efficient IR-style keyword search over relational databases. *In Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, September 9-12, 2003*, pages 850–861. Morgan Kaufmann, 2003.

[9] Andrey Balmin, Vagelis Hristidis et Yannis Papakonstantinou : Objectrank: Authority-based keyword search in databases. *In (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004*, pages 564–575. Morgan Kaufmann, 2004.

[10] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai et Hrishikesh Karambelkar : Bidirectional expansion for keyword search on graph databases. *In Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, page 505–516. VLDB Endowment, 2005.

[11] Vagelis Hristidis, Nick Koudas, Yannis Papakonstantinou et Divesh Srivastava : Keyword proximity search in XML trees. *IEEE Trans. Knowl. Data Eng.*, 18:525–539, 01 2006.

[12] Fang Liu, Clement Yu, Weiyi Meng et Abdur Chowdhury : Effective keyword search in relational databases. *In Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, page 563–574, New York, NY, USA, 2006. Association for Computing Machinery.

[13] Mayssam Sayyadian, Hieu LeKhac, AnHai Doan et Luis Gravano : Efficient keyword search across heterogeneous relational databases. pages 346–355. IEEE Computer Society, 2007.

[14] Yi Luo, Xuemin Lin, Wei Wang et Xiaofang Zhou : Spark: Top-k keyword query in relational databases. pages 115–126, 01 2007.

[15] Hao He, Haixun Wang, Jun Yang et Philip S. Yu : BLINKS: ranked keyword searches on graphs. *In Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 305–316, 2007.

[16] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang et X. Lin : Finding top-k min-cost connected trees in databases. *In ICDE*, 2007.

[17] Filipe de Sá Mesquita, Altigran Soares da Silva, Edleno Silva de Moura, Pável Calado et Alberto H. F. Laender : LABRADOR: efficiently publishing relational databases on the web by using keyword-based query interfaces. *Inf. Process. Manag.*, 43(4):983–1004, 2007.

[18] Guoliang Li, Beng Ooi, Jianhua Feng, Jianyong Wang et Lizhu Zhou : Ease: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. pages 903–914, 06 2008.

[19] Bhavana Bharat Dalvi, Meghana Kshirsagar et S. Sudarshan : Keyword search on external memory data graphs. *Proc. VLDB Endow.*, 1(1):1189–1204, 2008.

[20] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek et G. Weikum : Star: Steiner-tree approximation in relationship graphs. *In 2009 IEEE 25th International Conference on Data Engineering*, pages 868–879, 2009.

[21] Thanh Tran, Haofen Wang, Sebastian Rudolph et Philipp Cimiano : Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. *In 2009 IEEE 25th International Conference on Data Engineering*, pages 405–416, 2009.

[22] Joel Coffman et Alfred C. Weaver : Structured data retrieval using cover density ranking. *In Proceedings of the 2nd International Workshop on Keyword Search on Structured Data*, KEYS '10, New York, NY, USA, 2010. Association for Computing Machinery.

[23] Veli Bicer, Thanh Tran et Radoslav Nedkov : Ranking support for keyword search on structured data using relevance models. *In Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 1669–1678, New York, NY, USA, 2011. Association for Computing Machinery.

[24] Yi Luo, Wei Wang, Xuemin Lin, Xiaofang Zhou, Jianmin Wang et Kequi Li : Spark2: Top-k keyword query in relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 23(12):1763–1780, 2011.

[25] Andrey Gubichev et Thomas Neumann : Fast approximation of Steiner trees in large graphs. *In CIKM*, pages 1497–1501, 2012.

[26] Yosi Mass et Yehoshua Sagiv : Language models for keyword search over data graphs. *In Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, page 363–372, New York, NY, USA, 2012. Association for Computing Machinery.

[27] Wangchao Le, Feifei Li, Anastasios Kementsietsidis et Songyun Duan : Scalable keyword search on large RDF data. *IEEE Trans. Knowl. Data Eng.*, 26(11):2774–2788, 2014.

[28] Pericles de Oliveira, Altigran Soares da Silva, Edleno Silva de Moura et Rosiane Rodrigues : Match-based candidate network generation for keyword queries over relational databases. *In 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 1344–1347. IEEE Computer Society, 2018.

[29] Vinay M. S. et Jayant R. Haritsa : Root rank: A relational operator for KWS result ranking. *In* Raghu Krishnapuram et Parag Singla, éditeurs : *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, COMAD/CODS 2019, Kolkata, India, January 3-5, 2019*, pages 103–111. ACM, 2019.

[30] Vinay M. S. et Jayant R. Haritsa : Operator implementation of result set dependent KWS scoring functions. *Inf. Syst.*, 89:101465, 2020.

[31] Yuxuan Shi, Gong Cheng, Trung-Kien Tran, Evgeny Kharlamov et Yulin Shen : Efficient computation of semantically cohesive subgraphs for keyword-based knowledge graph exploration. *In The Web Conference*, 2021.

[32] Joel Coffman et Alfred C. Weaver : A framework for evaluating database keyword search strategies. *In Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, page 729–738, New York, NY, USA, 2010. Association for Computing Machinery.

[33] Naveen Garg, Goran Konjevod et R. Ravi : A polylogarithmic approximation algorithm for the group steiner tree problem. *In Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, page 253–259, USA, 1998. Society for Industrial and Applied Mathematics.

[34] A. Singhal : Modern information retrieval: a brief overview. *IEEE Data Engineering Bulletin, Special Issue on Text and Databases*, 24, 2001.

[35] Christopher D. Manning, Prabhakar Raghavan et Hinrich Schütze : *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.

[36] William Webber : Evaluating the effectiveness of keyword search. *IEEE Data Eng. Bull.*, 33:54–59, 2010.

[37] Angelos Christos Anadiotis, Oana Balalau, Catarina Conceiçao, Helena Galhardas, Mhd Yamen Haddad, Ioana Manolescu, Tayeb Merabti et Jingmao You : Graph integration of structured, semistructured and unstructured data for data journalism. *Elsevier Journal of Information Systems*, 2021.

[38] Angelos-Christos G. Anadiotis, Oana Balalau, Théo Bouganim, Francesco Chimienti, Helena Galhardas, Mhd Yamen Haddad, Stéphane Horel, Ioana Manolescu et Youssr Youssef : Empowering investigative journalism with graph-based heterogeneous data management. *CoRR (to appear in IEEE Data Engineering Bulletin)*, abs/2102.04141, 2021.

[39] Manish Singh, Michael J. Cafarella et H. V. Jagadish : DBExplorer: Exploratory search in databases. *In Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016*, pages 89–100. OpenProceedings.org, 2016.

[40] Haixun Wang et Charu C. Aggarwal : A survey of algorithms for keyword search on graph data. *In* Charu C. Aggarwal et Haixun Wang, éditeurs : *Managing and Mining Graph Data*, volume 40 de *Advances in Database Systems*, pages 249–273. Springer, 2010.

[41] Guoliang Li, Beng Ooi, Jianhua Feng, Jianyong Wang et Lizhu Zhou : Ease: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. pages 903–914, 06 2008.

[42] Joel Coffman et Alfred C. Weaver : An empirical performance evaluation of relational keyword search techniques. *IEEE Trans. Knowl. Data Eng.*, 26(1):30–42, 2014.

[43] Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, Marcin Sydow et Gerhard Weikum : Language-model-based ranking for queries on RDF-graphs. *In Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, page 977–986, New York, NY, USA, 2009. Association for Computing Machinery.