ConnectionLens: Finding Connections Across Heterogeneous Data Sources
Camille Chanial, Rédouane Dziri, Helena Galhardas, Julien Leblay,
Minh-Huong Le Nguyen, Ioana Manolescu

To cite this version:

HAL Id: hal-01841009
https://inria.hal.science/hal-01841009
Submitted on 17 Jul 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
ConnectionLens: Finding Connections Across Heterogeneous Data Sources

Camille Chanial  
Ecole polytechnique  
France  
camille.chanial@polytechnique.edu

Rédouane Dziri  
Ecole polytechnique  
France  
redouane.dziri@polytechnique.edu

Helena Galhardas  
INESC-ID and IST  
Univ. Lisboa, Portugal  
helena.galhardas@tecnico.ulisboa.pt

Julien Leblay  
AIRC, AIST  
Tokyo, Japan  
julien.leblay@aist.go.jp

Minh-Huong Le Nguyen  
Ecole polytechnique  
France  
minh-huong.le-nguyen@polytechnique.edu

Ioana Manolescu  
Inria and LIX, CNRS and  
Ecole Polytechnique, France  
iorea.manolescu@inria.fr

ABSTRACT

Nowadays, journalism is facilitated by the existence of large amounts of publicly available digital data sources. In particular, journalists can do investigative work, which typically consists on keyword-based searches over many heterogeneous, independently produced and dynamic data sources, to obtain useful, interconnecting and traceable information. We propose to demonstrate ConnectionLens, a system based on a novel algorithm for keyword search across heterogeneous data sources. Our demonstration scenarios are based on use cases suggested by journalists from the French journal Le Monde, with whom we collaborate.

1. MOTIVATION AND OUTLINE

The explosion of digital data sources on about every aspect of modern life has led journalists to collaborate with computer scientists, statisticians, and social scientists of various disciplines, aiming at novel digital tools to analyze and exploit this data. Since a seminal interdisciplinary study [2], the field of data journalism [10], that is, journalistic work significantly based on digital data, increasingly attracts the attention of journalists and computer scientists alike.

In this work, we focus on a core data journalism problem: identifying connections across a set of heterogeneous, independently produced data sources. We are inspired by investigative journalism work, which seeks out and explores connections between individuals, organizations, companies etc. Such work requires, first, getting access to data sources, through any means (from verbal or phone communication, to paper mail, fax, and any electronic transmission method); second, analyzing this data, to find out what valuable information it may contain. For instance, consider an example raised by journalists at Le Monde, a leading French national newspaper, with which we currently collaborate: France’s national elections in 2017 brought about an unprecedented ratio of first-time National Assembly elected members. Journalists scrambled to learn as much as possible about the nation’s new representatives, in particular asking e.g., “what connections the new representatives have (or have had) with companies?” This question is interesting to identify areas of economic competence, but also possible conflicts of interest. By law, French representatives must disclose direct financial interests, but more indirect connections (e.g., being, for many years, a close collaborator of a company’s current CEO) must be dug out by the press.

Many digital data sources which could be used to answer such questions are publicly available today. However, finding answers to journalists’ queries is challenging for many reasons. (i) The data is large, precluding the use of the hand-crafted tools (often spreadsheets) that journalists are used to; (ii) there are many, structurally heterogeneous, independently-produced data sources; some sources, e.g. national company registry, are relational, some others, e.g. contracts, discourses are text, social media content comes as JSON documents, open data is often structured in RDF graphs etc.; (iii) an answer stating, e.g., that a certain representative has studied with the CEO of a given company, may require interconnecting information from several data sources, e.g., the history of company C, the Wikipedia page of A, and the public information available on D and its CEO, namely B; (iv) journalists do not know the shape, size, and structure of the connections they are seeking out, thus they need the ability to search through keywords; (v) the set of data sources is highly dynamic, as journalists collect various data sources and try to see what insights they can get by combining them with existing ones. It is also worth stressing that newsrooms function under extreme time constraints, making it unfeasible to clean, consolidate and integrate all data sources in a unified warehouse. Finally, (vi) a staple of professional journalism is to be able to show evidence for a published claim. Thus, in an answer such as the one outlined in (iii) above, it is important to be able to show where each piece of information came from and how the connections were made.

We have built and propose to demonstrate ConnectionLens, a system addressing the above challenges; at its core is a novel algorithm for keyword search across a set \( \mathcal{D} \) of heterogeneous data sources, each of which can be: a relational source; a set of keywords; a JSON document; a text file; or an RDF graph.

For instance, Figure 1 shows a JSON datasource \( DS_1 \) containing information about elected representatives, a text dataset \( DS_2 \) listing the alumni of École polytechnique, where many French company executives have studied, and a relational source \( DS_3 \) providing information about companies and their CEOs. A query \( Q \) is a set of keywords...
{\omega_1, \ldots, \omega_n}\) for some integer \(n \geq 1\). In Figure 1, the query “En Marche company” seeks to find out connections between elected representatives from the “En Marche” political party, and some company. An answer tree to \(Q\) over \(D\) is a tree, part of a virtual graph (which, as we explain later, is the way we view \(D\)). Each answer tree node \(n_i\) comes from a dataset \(D_i \in \mathcal{D}\), and each edge \(e_{ij}\) either comes from a dataset \(D_j \in \mathcal{D}\), or is a link between two nodes (from the same or from two different datasets) whose data content we find sufficiently similar to consider them “the same” for the purpose of our querying. Further, each node corresponding to an occurrence \(\lambda\) attaches to each node its URI or literal (constant) label in \(D\). The property labeling every edge \(e\) carries a text label (we use \(\lambda\) also to denote the edge labeling function) and a confidence \(c\), which is between 0 and 1, whose usage will be shortly explained below. Node and edge labels appear as quoted strings in Figure 1. Based on this figure, we explain below how \(\mathcal{G}\) nodes and edges are (conceptually!) derived from each data source \(D\).

\section{2. VIRTUAL GRAPH}

To be able to exploit data connections occurring within and across all datasets \(D \in \mathcal{D}\), we view all data sources as a single, virtual graph \(\mathcal{G}\), and answer keyword search queries with subtrees of this graph. Each \(\mathcal{G}\) node has a (globally unique) identifier, some of which are shown in Figure 1 in blue. A label function \(\lambda\) assigns to every node a (possibly empty) text label, reflecting its content in \(D\). \(\mathcal{G}\) edges are directed; each edge \(e\) carries a text label (we use \(\lambda\) also to denote the edge labeling function) and a confidence \(c\), which is between 0 and 1, whose usage will be shortly explained below. Node and edge labels appear as quoted strings in Figure 1. Based on this figure, we explain below how \(\mathcal{G}\) nodes and edges are (conceptually!) derived from each data source \(D\).

\subsection{2.1 Nodes and edges derived from a source}

First, a dataset node \(n_D\) is created in \(\mathcal{G}\) to represent \(D\) itself, e.g., \(n_{DS1}\) to \(n_{DS3}\). Every \(\mathcal{G}\) node corresponding to data from \(D\) (see below) is connected to \(n_D\), through an edge labeled \(\text{origDS}\) (standing for originating data source); we show such edges with dotted lines in the figure. They ensure that any two virtual graph nodes coming from the same data source \(D\) are connected at least through \(n_D\).

The other nodes and edges of \(\mathcal{G}\) are defined as follows:

(i) If \(D\) is a relational database such as \(DS_1\), \(\mathcal{G}\) has a node for each constant, list and map occurring in \(D\), and an edge labeled \(\text{origDS}\) connects the node representing \(D\) to the one corresponding to the top list or map in \(D\). For each \((n_1, v_1)\) name-value pair in a map, \(n_1\) becomes the label of the edge leading to the node corresponding to \(v_1\).

(ii) If \(D\) is a text such as \(DS_2\), we apply entity and relationship extraction to identify in \(D\) occurrences of entities (such as people, places, organizations etc.) and of relationships (such as bornIn, worksFor etc.) Any off-the-shelf extractor (or set of extractors) can be used; CONNECTIONLENS currently uses OpenCalais (http://www.opencalais.com) for entity extraction. A \(\mathcal{G}\) node is created to each extracted entity (resp. relationship) occurrence; its \(\lambda\) label is the exact text snippet identified by the extractor; it has a type edge pointing to the entity type identified by the extractor, e.g., OC:Person stands for OpenCalais’ Person type URI, and child nodes containing the offset and length of its appearance in the original text (omitted in Figure 1 to avoid clutter). Further, each node corresponding to an occurrence of a relationship between two entities, is connected to the nodes corresponding to the respective entity occurrences by edges identifying the entity roles in the relationship.

(iv) If \(D\) is a relational database such as \(DS_3\), for each relation \(R(a_1, a_2, \ldots)\) in \(D\) and each tuple \(r \in R\), \(\mathcal{G}\) contains a node \(n_r\), with outgoing edges labeled \(a_1, a_2\) etc. toward \(\mathcal{G}\) nodes, labeled with the values of the respective attributes of \(r\). The label of \(n_r\) is one of its primary keys (we add such a primary key attribute if \(R\) doesn’t have one). Further,
for any two relations $S,T \in D$ such that $S.a$ is a foreign key corresponding to the primary key $T.b$, and tuples $s \in S, t \in T$ such that $s.a = t.b$, $G$ comprises an edge $n_s \rightarrow n_t$. For instance, if $S.spouse$ is a foreign key on $T.id$, then $G$ comprises $n_s \rightarrow n_t$. This graph view of a relational database is often used for keyword search, e.g. \cite{12,17}.

(v) Any $G$ edge whose label $\lambda(n)$ is longer than a threshold $\theta_{\text{cut}}$ is treated like a text data source, i.e., entity and relationship occurrences are extracted as in (iii); however, the $G$ nodes created from these occurrences are all descendants of $n$, and their original data source is that of $n$. This both provides a uniform treatment of text regardless of where it appears, and records the origin of each text content.

All virtual $G$ edges described above have a confidence of 1.0. Some extractors provide a confidence value for the extraction: this can be attached to the edges between the nodes, e.g. nDS1.V1, and their types, e.g. OC:Person.

### 2.2 SameAs edges

We add to $G$ an edge labeled sameAs between nodes $n_1, n_2$ (from the same or from different data sources), as soon as they are similar beyond a certain threshold $\theta_{\text{sim}}$; the confidence of such an edge is the similarity score, normalized to $[0,1]$. Identifying when two data objects represent the same thing (aka approximate duplicate detection, etc.) is a thoroughly studied data integration problem \cite{9}, for which many solutions exist, especially among similarly-structured objects, e.g. \cite{11,18}. CONNECTIONLSENS interconnects disparate sources that may not even have the same data model, e.g., an extracted entity may match with a JSON value (e.g., “Anne Martin”), or with an attribute in a relational tuple, e.g. “Philippe Varin” and “P. Varin”; or, an interesting link may exist between, say, a person and a company, if they both mention a certain user in their tweets etc. We seek to find all such value connections, and distinguish the trivial from the interesting ones using a score (Section 4).

CONNECTIONLSENS uses the labels $\lambda(n_1), \lambda(n_2)$ of two $G$ nodes to decide if they are the same. If the labels are shorter than a certain size limit $L$, the Jaro distance between them is compared with $\theta_{\text{sim}}$. If $\lambda(n_1)$ or $\lambda(n_2)$ are longer than $L$, we turn them both into bags of words and then compute their set-based Jaccard distance. If $\lambda(n_1), \lambda(n_2)$ are identical URLs, we connect them through sameAs with a confidence of 1.0. Different distance functions or comparisons can be plugged in as users get familiar with data sources, in pay-as-you-go data integration fashion \cite{3}.

We do not build sameAs links based on nodes’ adjacent edges and neighbor nodes. However, sameAs links based on labels alone, e.g., a common last name of two people, suffice to establish the data connections we are interested in.

### 2.3 Virtual graph indexing

We encode, index and store the $G$ nodes and edges derived from a source $D$ in a set of data structures, as follows. 1. We compute an ID $id_D$ for the dataset.

2. We derive nodes and edges as explained above from $D$. For each node $n$, we compute an ID $id_n$ prefixed with $id_D$. This de facto encodes the edge $n_D \xrightarrow{\text{origD}} n \xleftarrow{\text{origD}} id_n$. 3. We compute $\lambda(n)$ from the original text content of $n$, through stop word and punctuation removal, and stemming. 4. For each word $w \in \lambda(n)$, we insert $(w, id_n)$ in the index $I(\text{word, node})$. Similarly, $\lambda(e)$ is computed for each edge $e$ and its words indexed in $I$.

5. We seek to find sameAs edges to which a node $n$ derived from $D$ may participate. We look up all the nodes $n'$ whose labels share at least a word with $n$, that is, $(w, id_n)$ and $(w, id_{n'})$ belong to $I$, and compute the similarity between $\lambda(n)$ and $\lambda(n')$. All sameAs edges are stored in a bridge table $B(id_1, id_2, c)$ which records that the nodes identified by $n_1, n_2$ are judged the same with confidence $c$. For instance, the $B(nDS1.V1, nDS3.V2, 0.76)$ tuple encodes the sameAs link between the two nodes corresponding to Philippe Varin in Figure 1.

### 3. ANSWERING KEYWORD QUERIES

Given a query $Q = \{w_1, \ldots, w_k\}$, the problem of finding the $k$ answer trees (recall their definition in Section 1) we call them ATs, in short) with the best score is known to be NP-hard, by reduction to the Steiner tree problem. Thus, CONNECTIONLSENS adopts a heuristic method to enumerate answer trees and returns the $k$ highest-score ones.

The AT enumeration algorithm starts by looking up in the index $I$ for the potentially interesting data sources for $Q$, denoted $P(Q)$, that is: the data sources from which $G$ nodes matching some query keywords are derived. For each sub-query $Q'$ of $Q$, and each source $D \in P(Q)$ containing exactly the keyword set $Q'$, the procedure localSearch$(D, Q')$ returns the ATs (if any) whose nodes and edges derive only from $D$. The implementation of localSearch$(D, Q)$ depends on the nature of $D$: it follows the lines of \cite{12} for relational sources, \cite{1} for JSON, and \cite{13} for RDF data. The algorithm starts by optimistically asking each $D \in P(Q)$ for ATs for the largest subset of $Q$ for which $D$ has matches. If $D$ has only one connected component, it is sure to contain at one such AT; also note that ATs are undirected, that is, $G$ edges form an AT as soon as they share a node, regardless of the direction of the edges. For instance, an AT may consist of three $G$ edges $n_1 \xrightarrow{a} n_2 \xrightarrow{b} n_3 \xrightarrow{c} n_4$. This is because we are interested in data connections, and find an edge such as $n_1 \xrightarrow{a} n_2$ interesting as a link, in both directions.

Partial ATs (initially, those local to each data source) are inserted in a priority queue $U$, based on their score (discussed below). The algorithm greedily picks the top-score AT $t$ from $U$, and adds it to the result set if it is an answer to $Q$ and its score is among the $k$ best so far. If $t$ is just a partial answer, we try to find another partial tree $t'$ to combine with $t$, through a sameAs edge between a node from $t$ and one from $t'$. The AT $t''$ resulting from $t$ and $t'$ is again added to $U$ and the process continues, until a time-out occurs or there are no ATs left to add in $U$. If answers to $Q$ are not found after a certain time, localSearch calls are made with smaller Q subqueries, in the hope that the resulting ATs may participate to more sameAs edges allowing to combine them in answers to $Q$.

The score $s(t)$ of an AT $t$ comprises for each $w_i \in Q$, a matching score $ms(s_i(w_i))$ reflecting the extent to which the labels of all $t$ nodes and edges match $w_i$, as well as a structure score $\xi(t)$ depending on the AT structure. We quickly found that unlike many previous keyword search works, small ATs are not always preferable because they may bring trivial (uninteresting) information. For instance, any French representative may be connected to any French company through a node labeled “France” (where they reside). Instead, for our journalistic applications, we favor a specificity metric, computed as follows. The edge $n_1 \xrightarrow{a} n_2$ is specific in an AT, if $n_1$ has few outgoing a edges, and $n_2$ has few incoming a edges. Then, $\xi(t)$ is a weighted sum of the
average specificity of its edges; and the product of its edge scores for strictly more keywords than \(ms\); otherwise, a weighted combination of the \(ms\) and \(\xi\) values for both trees is used to decide who has the highest score.

4. IMPLEMENTATION AND SCENARIOS

ConnectionLens is developed in Java and Python, relying on Postgres and Jena to store the data sources. We implemented localSearch for relational, JSON, (annotated) text and RDF data, with the rest of the algorithm from Section 3. We plan to show two scenarios.

Scenario 1 is based on (i) JSON data from the Regards Citoyens NGO that publishes all the French parliament activity, including all National Assembly representatives (800KB); (ii) wikitidata information about people, for example their past jobs (1.55GB JSON file); (iii) a text dump of recent articles published in main French media; and (iv) a text dump of Journal Officiel, France’s official public law repository. Starting from political parties and companies, e.g., {“En Marche”, “Areva”} from our motivating example, we will search for connections between them using the information available from the underlying data sources.

Scenario 2 Here, we aim at identifying political leaders who spread journalistic hoaxes. We will use: (i) a DBPedia RDF graph of French political leaders; (ii) a JSON document containing tweets from French political leaders; (iii) a known JSON database of hoaxes, provided by Le Monde. Given queries such as {“Front National”, “Macron”, “hoaxes”}, we will investigate connections between pairs of political leaders through hoaxes, e.g., who disseminates on the social media hoaxes about the other one.

Figure 2 shows a screenshot of the possible answers to the initial keyword-based query.

5. RELATED WORK

Our work is related to several areas of prior research. Large knowledge bases such as DBPedia or Yago are currently available, yet journalists find “blind spots” in these ones, as relatively little-known people, events etc. are not covered there, yet they are described in details in other specific and/or local data sources. ConnectionLens is meant exactly to enable using together data sources of various nature, model, format etc. Exploiting together large data source collection is a core goal of data integration systems, and in particular dataspaces [9] or more recently data lakes [4]; our work can be seen as keyword search for dataspaces, following [7]; in contrast with that work, we seek to find connections across distinct data sources, whereas they only find local answers. Recent works focused on finding how to join large tables sets, based on their schema and sets of values in each column [5]. In contrast, we (also) work with semi-structured or text data, and aim at identifying connections (whose shape, size, and organization are unknown to the user asking a keyword query) among items scattered across a set of data sources. Keyword search is well-studied in text databases; more recently, algorithms have been proposed to answer keyword queries in structured databases including relational [12] RDF [8] ones; in all these works, each keyword query answer is local to one data source. Active learning is used in [16] to help integrate and exploit heterogeneous relational databases through keyword queries, whereas we consider more varied data sources, and seek to find data connections across them, also with a focus on answer specificity.

6. REFERENCES