# Toward Visual Interactive Exploration of Heterogeneous Graphs

Irène Burger, Ioana Manolescu, Emmanuel Pietriga, Fabian Suchanek

# Toward Visual Interactive Exploration of Heterogeneous Graphs


Irène Burger
irene.burger@polytechnique.edu
Institut Polytechnique de Paris, Inria
Palaiseau, France

Ioana Manolescu
ioana.manolescu@inria.fr
Inria, Institut Polytechnique de Paris
Palaiseau, France

Emmanuel Pietriga
emmanuel.pietriga@inria.fr
Univ. Paris-Saclay, CNRS, Inria
Orsay, France

Fabian Suchanek
suchanek@telecom-paris.fr
Télécom Paris, Institut Polytechnique de Paris
Palaiseau, France



## ABSTRACT

An interesting class of heterogeneous datasets, encountered for instance in data journalism applications, results from the interconnection of data sources of different data models, ranging from very structured (e.g., relational or graphs) to semistructured (e.g., JSON, HTML, XML) to completely unstructured (text). Such heterogeneous graphs can be exploited e.g., by keyword search, to uncover connection between search keywords [1].

In this paper, we present a vision toward making such graphs easily comprehensible by human users, such as journalists seeking to understand and explore them. Our proposal is twofold: (*i*) *abstracting* the graph by recognizing structured entities; this simplifies the graph without information loss; (*ii*) relying on *data visualization* techniques to help users grasp the graph contents. Our work in this area continues; we present preliminary encouraging results.


## 1 INTRODUCTION

Data journalists often have to handle sets of different data structures, which they obtain from official organizations or from their sources, extract from social media, receive via email or create themselves (typically Excel or Word-style) etc. For instance, journalists from the Le Monde newspaper want to retrieve *connections between elected people at Assemblée Nationale and companies that have outposts outside of France*; such a query can be answered currently at a high human effort cost, by inspecting e.g. a JSON list of Assemblée elected officials (available from NosDeputes.fr) and manually connecting the names with those found in a national registry of companies. This huge effort may still miss connections that could be found if one added information about politicians' and business people's spouses, information sometimes available in public knowledge bases such as DBpedia, or in a journalists' personal notes.

**ConnectionLens heterogeneous graphs.** The ConnectionLens project [1] aims to enable journalists to *work with data sources as they come*, and *quickly* due to the speed of the news cycle. This precludes the time to understand, analyze, and extract the data into a single unified data model. Instead, in ConnectionLens we consider that *the data model of a given dataset is "an accident" related to its creation history, and should not be a barrier toward exploiting it.*

As no single query language can be used on such heterogeneous data, ConnectionLens supports *keyword queries*, asking for all the connections that exist, in one or across several data sources, between these keywords. This problem has been raised by our collaboration with Les Décodeurs, Le Monde's fact-checking team, with whom we collaborate within the ContentCheck research project. The novelty of ConnectionLens search wrt the literature on keyword search in databases is to seek answers which may span over *several datasets of different data models*, with very different or even absent internal structure (the latter is true for text data).

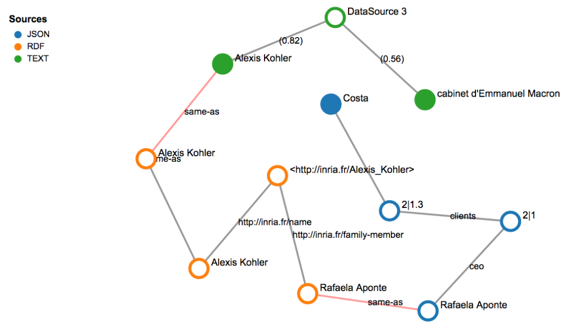


**Figure 1: Sample screen shot of ConnectionLens GUI [1] showing an answer to a three-keyword query.**

For instance, Figure 1 shows an answer to the three-keyword query: {"Macron", "Kohler", "Costa"}. Different colors indicate nodes from data sources of different data models (JSON, RDF and text, respectively); the nodes filled with solid color are those that match the query keywords. Nodes are either derived from the original dataset, e.g., a node for each tuple and attribute from a relational dataset, a node for each map or list in JSON, for each node in an RDF graph etc., or they may be *entity occurrences*, identified by a Named Entity Recognition module capable for now to identify People, Organizations and Locations. We extract entity occurrences from any text, whether a document or a phrase or a name appearing in a JSON node. The dataset interconnection is materialized by the two red edges labeled same-as between pairs of entity occurrences.

**The problem: ConnectionLens graph exploration.** While such a raw node-link diagram visualization allows users to find some results in heterogeneous ConnectionLens graphs, the support they provide for *exploring and understanding* the graph to non-expert users, such as journalists, is quite limited.

**(1.)** The visualization only shows the answer tree; the full graph is much larger, and a simple *node-link diagram* as this one does not scale beyond a dozen edges, as illustrated on a ConnectionLens graph[1] of barely more than 200 edges. *This problem is common to any large data graph* (not specific to ConnectionLens) [6]. Many techniques have been developed to address this issue, going from a simple navigable node-centric GUI[2] for a large RDF knowledge base [7], to more elaborate visualization techniques, e.g., [4].

(**2.**) While some ConnectionLens nodes are meaningful for human users, e.g., the entity occurrences recognized in Figure 1, others are not, as they merely correspond to *internal containers*, such as JSON maps or arrays, relational tuples etc., which do not carry human-understandable significance; examples are the nodes labeled 2|1 and 2|1.3 in the figure. Showing these nodes at the same level, therefore, is not appropriate. *This problem is specific to ConnectionLens*: it originates in its very goal of connecting information, no matter what format it comes from.

In this paper, we outline a vision for dramatically increasing the usability of ConnectionLens graphs, in particular through novel interactive visualizations based on recent advances in the area of expressive node-link diagram authoring [5] for multivariate networks [3]. We identified two steps toward realizing that vision: graph abstraction (Section 2), and graph visualization (Section 3).We present some preliminary results, and perspectives for our work.

## 2 ABSTRACTING CONNECTIONLENS GRAPHS

We present our analysis of the ways in which ConnectionLens graphs can be simplified, or abstracted, based on a set of observations made by journalists (from Le Monde and TF1) and data journalists / data visualization professionals (from WeDoData, a French company) with whom we shared them.

### 2.1 Abstraction Principles

We present a set of guiding principles, which we identify by a capital letter to be able to refer to them in the paper.

(*A*) *Entities are interesting.* Users want to know who, or what, a given dataset, or multi-dataset graph, is about. It is natural to be interested in people (e.g., a politician or a businessperson), organizations (e.g., an army or a company), or a place (e.g., the city where one lives, or a country such as Panama). Depending on the dataset and the application, of course, other kinds of entities may be considered, e.g., a Web site, a Facebook or Twitter account, a specific kind of organization (e.g., companies from Panama) etc.

(*B*) *Datasets may or may not be interesting.* Conceptually, we like to think of "data" as an abstract set of information, say, a large graph. From a practical viewpoint, however, data comes in datasets, which delimit "original subsets" of the global graph. The contours of a dataset are sometimes clear, e.g., a tweet, a Web page, or a speech; in other cases, they are more fuzzy. For instance, if a relational database holds three tables, should we view this as one or three datasets? From a user perspective, one can choose to ignore the datasets (make their boundaries invisible); this may be appropriate, for instance, if in a social network graph, we want to focus just on connections between users and/or hash tags. Alternatively, datasets can play a very significant role: (*i*) if entities co-occurring in a dataset denote an interesting association, e.g., two recipients of the same email; or (*ii*) if we identify connections between users and datasets, e.g., user Alice *hasAuthored* dataset article1, on which user Bob *commented* etc.

(*C*) *Containers are rather uninteresting.* Here, containers denote: a table (or set of tuples) if the data is relational, or a JSON array. Intuitively, containers serve to group together several more-or-less comparable "things", such as albums of a singer, or members of a committee. The container node itself is less useful.

(*D*) *Rich entity nodes are desirable for data exploration.* Here, an entity node (EN, in short) denotes an instance of an Entity, in the traditional Entity-Relationship sense known from conceptual design. For instance, "the person François Ruffin, having the birthplace Amiens and the twitter handle @François_Ruffin" is an EN. Observe that *an EN is a larger and more complex notion than an entity occurrence currently identified by the extraction*; in particular, an entity occurrence is always a leaf (added as a child of the text node where the extractor found it), and has no attributes.

As customary in the Entity-Relationship model, we assume that some of the nodes surrounding an EN *n*, and reachable from *n*, e.g., the twitter handle above, only serve to describe *n* and are not standalone nodes. However, we depart from the classical notion of an Entity used in relational database design, by allowing ENs of a given type to have different sets of attributes, and/or to have more than one value for a given attribute. Note also that in some datasets, Amiens may also be an EN, e.g., if the dataset specifies things about it such as its population, geographical coordinates etc.; in other datasets, e.g., one centered around individuals, or around shops distributed across a country, Amiens may be just a dimension characterizing the ENs (people, respectively, shops). We believe ENs are useful paradigms for exploring the dataset because:

- They correspond to a natural paradigm of "things" characterized by their properties (attributes);
- They group together pieces of information related to a common resource, e.g., the twitter and Facebook handles of a given person;
- They lay the foundation for many interesting visualizations, where attributes can be used e.g. to place shops on a map according to their location, or events on a timeline etc.

(*E*) *Relationships between ENs are interesting.* Here, we extend the intuition that just like ENs, relationships are intuitive constructs that allow to structure and analyze a CL graph. For instance, it is interesting to find that Alice (an EN) *supervised* Bob (another EN).

Based on the above principles, below we describe an algorithmic approach which, starting from a ConnectionLens graph, (1) identifies Entity Nodes, (2) assigns them attributes among the nodes in their neighborhood, and (3) connects them through relationships.

### 2.2 Abstraction Algorithm

As an example, Figure 2 shows a ConnectionLens graph resulting from a JSON document describing Nobel laureates. The red node is the dataset; blue nodes are maps or arrays, while green nodes correspond to values (literals). Further, named entity occurrences identified by the extraction are outlined by a black contour, e.g., Jean S., Inria, CNRS etc. These form the basis of the entity nodes we want to identify. The figure also shows that ConnectionLens creates a single node for all the nodes with the same label found on the same root-to-leaf path in a dataset: thus, there is a single

[1] http://pages.saclay.inria.fr/ioana.manolescu/DOT-obtained-image.pdf
[2] https://gate.d5.mpi-inf.mpg.de/webyago3spotlx/SvgBrowser

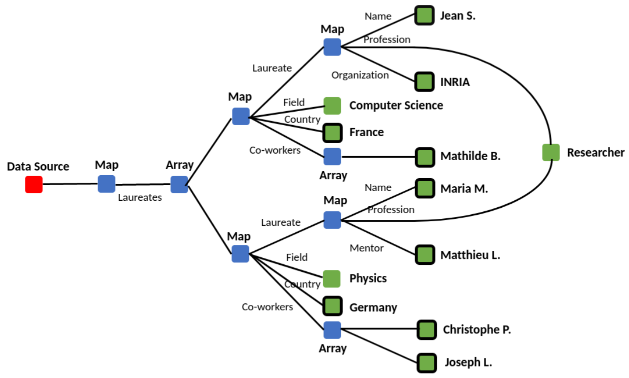


**Figure 2: Input graph $G(d)$ from a JSON document.**

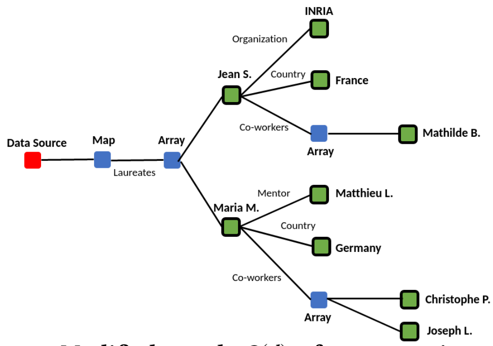


**Figure 3: Modified graph $G(d)$ after extracting entities from the graph in Figure 2.**

node labeled "Researcher". This decision materializes the connection which exists between Jean S. and Maria M.: both are researchers.

Given a graph $G(d)$ such as shown in the figure, corresponding to a dataset $d$, a graph $EG(d)$ consisting of entity nodes and relationships between them can be built as follows.

(1) Following $(A)$ and $(D)$, we seek to create ENs starting from the entity occurrences. To that purpose, we use a priority queue $Q$ in which we push all pairs (entity occurrence, its parent node) from $G(d)$; the priority is computed as the length of the path from the occurrence to the parent until the dataset root (the longer the path, the higher the pair's priority). If an entity occurrence has several parents, it is pushed in $Q$ once for each parent.

(2) While $Q$ is not empty: Pop from $Q$ the pair $(o, p)$ with the highest priority. The *type* of $o$, denoted $\tau(o)$, is available in the graph $G(d)$; the set of entity occurrence types currently supported is $\mathcal{T}$ = {Person, Location, Organization}. We need to create a rich entity node $EN(o)$ out of $o$. As $o$ is a leaf node (created by ConnectionLens extraction), to find possible attributes of $EN(o)$, we need to climb up from $o$ one or a few levels, then go down to find $o$'s attributes in the vicinity.

   (a) If $p$ is an array (list), e.g., in the case when $o$ is the entity occurrence "Mathilde B.", we conclude that $o$ has no attributes among its siblings, as one does not expect to find, in an array, entities *and* attributes of the same (or comparable) entities.

   (b) If $p$ is a map, e.g., when $o$ is the entity occurrence "Jean S.", we find the first edge with a non-empty label $\lambda$ on the path from $p$ to the dataset node; in our example, $\lambda$ is "Laureate". We make the assumption that $\lambda$ carries useful information about the content (meaning) of the map $p$.

   Next, we need to understand how $p$ relates to the entity node $EN(o)$ we are trying to build. For each type $\tau \in \mathcal{T}$, we compare $\lambda$, using Embedded Word Distances [2], to a manually chosen set of keywords $W_\tau$, and select the type $\tau_\lambda$ to which $\lambda$ is closest. When $\lambda$ is "Laureates", $\tau_\lambda$ is Person. Then:

   - If $\tau_\lambda \neq \tau(o)$, the parent $p$ comprising $o$ is "not about $o$"; $p$ likely describes something else. In our example, where $\tau(o)$ is Person, if $\tau_\lambda$ is Organization, $p$ may describe, e.g., an organization in which $o$ plays some role, but not $o$ itself, therefore the siblings of $o$ are not its attributes. In this case, from $o$, we can find no more attributes of $EN(o)$.
   - On the contrary, if $\tau_\lambda = \tau(o)$, we consider that the $p$ *may be about* $o$, and try to find $p$ children (siblings of $o$) to attach to $EN(o)$ as attributes. For that purpose, we examine all the entity children $c$ of $p$ with type $\tau(c) = \tau_\lambda$. If $o$ is the only one, as is "Jean S.", then its siblings that are leaf children of $p$ and are not entities themselves are considered as attributes of $EN(o)$. Otherwise, e.g., "Maria M." has a sibling "Matthieu L." which is also of type $\tau$ = PERSON, then for each edge $p \xrightarrow{a} c$, where $c$ is the child of $p$ with type $\tau_\lambda$ and $a$ is the label of the edge, we compare $a$ to a manually chosen set of keywords whose meaning is similar to "key", e.g., "ID", "name" etc. The label $a$ with the smallest distance to this set determines the "winner" entity occurrence (child of $p$) which captures its siblings as attributes. In our example, "Name" is closer in meaning to "key" than "Mentor", so "Maria M." is chosen. In application of our principle $(C)$, when $EN(o)$ captures an attribute, $EN(o)$ replaces its container parent $p$ in $G(d)$, and is pushed back in $Q$ paired with $p$'s parent.

   (c) If $p$ is a JSON value from which several entities occurrences were extracted (e.g., $p$ is a long text, say, a politician's speech), no attribute of $EN(o)$ is extracted from $p$.

This algorithm creates a set of ENs, each encompassing several interconnected nodes from the original graph $G(d)$. It also modifies the structure of $G(d)$ as shown in Figure 3.

Finally, to satisfy principle $(E)$, we look for *relationships (links)* to be added to $EG$, based on paths found in the modified graph $G$ as shown above. For now we consider two simple approaches;

- If the shortest path between two ENs in $G$ has a length below a fixed threshold, we create an edge between them in $EG$, labeled with the concatenation of labels on this path. For example we create the edge "Co-workers" between "Maria M." and "Joseph L."
- If two ENs have identically labeled paths to their nearest common ancestor *nca*, we create an edge between them, whose label is *"co-"* concatenated with the sequence of labels on this path. If *nca* happens to be an array, we add to this the first label encountered on the path from *nca* to the datasource.

Finally, to avoid overloading $EG$ with edges, we do not create an edge between two entities if the shortest path between them in $G$ goes through another EN. Figure 4 shows the resulting graph.

The above algorithm handles *one* dataset, a *JSON* one. It directly applies to *relational* data (where a tuple plays the role of a map and a relation that of an array), *(X)HTML* documents, and *text* documents which we view as shallow trees (one dataset node with entity occurrence children). Further, two ENs $e_1, e_2$ extracted from datasets $d_1, d_2$ such that the originating entity occurrences $o_1, o_2$ were connected by same-As in $G(D)$ are unified in the final graph.

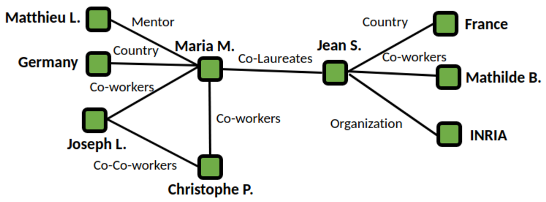


**Figure 4: Graph of extracted entities *EG* from example**

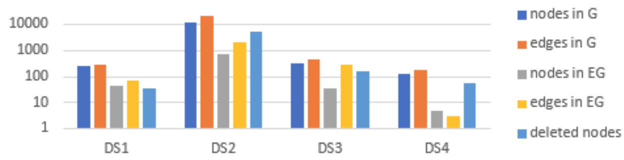


**Figure 5: Impact of the graph abstraction algorithms.**

## 3 VISUALIZING ENTITY GRAPHS

As mentioned earlier, node-link diagram representations of graphs do not scale beyond a few dozen nodes and edges. Alternative representations of graphs exist, such as adjacency matrices, but are much less familiar to users and require some training to interpret. Furthermore, adjacency matrices complicate tasks that involve following paths in graphs, which is an essential aspect of analysis in ConnectionLens. Thus, we explore *ways to support the visual exploration of ConnectionLens graphs by improving the more familiar node-link diagram representation*, addressing problems of scalability both in terms of number of nodes and links, and in terms of attributes of the multivariate network.

The abstraction strategy we explored so far (Section 2) can be seen as a pre-processing before generating a visual representation of the graph. Another strategy would consist in converting some links in the graph that point to leaf nodes (literal values) into attributes of the source node. Conceptually similar to what GSS [4] does for RDF graphs, applying such a strategy means that the node attributes that have been preprocessed this way can then be visually conveyed by encoding their value using visual variables of the nodes themselves, as is typically done in multivariate network visualization [3]. Then, expressive node-link diagram authoring environments such as Graphies [5], which feature a rich palette of visual mapping options, will let users create elaborate visual representations in which nominal, ordered and quantitative attributes can be conveyed using different properties of the graphical nodes: shape (predefined shapes or sketches), size, fill and stroke color hue/saturation/brightness, label font properties, even position depending on the choice of layout strategy.

The second strategy consists in incrementally building the graph based on one or a few named entities of interest, iteratively adding subsets of nodes and relationships based on their type or other criterion (e.g., a threshold value for a given attribute), specified interactively by the user in the visualization environment. Thus, users populate the initially-empty canvas with nodes and links of particular interest, adding/removing elements by direct manipulation. Combined with a history mechanism that enables backtracking and forking to explore different subsets of the original ConnectionLens graph, this bottom-up approach to building the visual representation makes it possible to explore graphs that would not be amenable to visualization considered in their entirety. Here again, we rely on Graphies [5] to enable such an incremental construction of the graph. While Graphies is fully functional, we are still in the early stages of integrating it with ConnectionLens. Preliminary results, that rely on a manual processing pipeline for now, are encouraging.

## 4 PRELIMINARY EXPERIMENTS

We implemented in Python the algorithm outlined previously. We present some quantitative results based on tests made over four heterogeneous datasets denoted $DS1$, $DS2$, $DS3$ and $DS4$. $DS1$ is composed of four datasets : two JSON documents describing National Assembly member and a tweet from F. Ruffin respectively, as well as two texts. The resulting ConnectionLens graph (the one in[1]) is not easy to understand; it has many structural nodes and also uninteresting metadata part of the Tweet content. $DS2$ is a large dataset representing a list of Nobel Laureates. This comprises many Person and Location entity occurrences, as well as many attributes: birth dates, professions etc. $DS3$ is a RDF graph describing French politicians. $DS4$ describes the career of one politician, most nodes describe him, while other entities are cities where he was elected.

Figure 5 shows, for each dataset, the number of edges and nodes in $G(d)$, respectively, $EG(d)$. We see that graph abstraction brings about an order of magnitude reduction in the number of nodes and edges. The “deleted nodes” are neither recognized ENs nor attributes of one. A text with no entity occurrence, or children of a map or array which does not contain entities, is not present in $EG$, nor in the visualization.

## 5 PERSPECTIVES

Attribute assignment needs to be refined: we currently assume a map refers to at most one EN. However, a map can describe several ENs, or a link between several ENs, in which case the attributes should be added to the link instead of the ENs. For example, if we consider a contract, the id of the contract should be an attribute of the edge link the ENs that are part of this contract. We aim to improve our work in order to take these occurences into account.

To improve the visualization, we also seek to simplify the labels of the edges between entities. Indeed, in large graphs, concatenating labels found on a path will lead to long and unintelligible labels. One idea is to simplify these labels by recognizing a more general type of link such as "worksIn", "writesAbout", "worksWith", etc.

**Acknowledgements.** This work was partially supported by ANR-15-CE23-0025 (ContentCheck) and ANR-16-CE23-0007 (DICOS).

## REFERENCES


[1] Camille Chanial, Rédouane Dziri, Helena Galhardas, Julien Leblay, Minh Huong Le Nguyen, and Ioana Manolescu. 2018. ConnectionLens: Finding Connections Across Heterogeneous Data Sources (demonstration). *VLDB* (2018).

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv* (2013).

[3] C. Nobre, M. Meyer, M. Streit, and A. Lex. 2019. The State of the Art in Visualizing Multivariate Networks. *Computer Graphics Forum* 38, 3 (2019).

[4] E. Pietriga. 2006. Semantic Web Data Visualization with Graph Style Sheets. In *SoftVis*.

[5] Hugo Romat, Caroline Appert, and Emmanuel Pietriga. 2019. Expressive Authoring of Node-Link Diagrams with Graphies. *TVCG* (2019).

[6] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J.J. van Wijk, J.-D. Fekete, and D.W. Fellner. 2011. Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges. *Computer Graphics Forum* 30, 6 (2011).

[7] Gerhard Weikum, Johannes Hoffart, and Fabian M. Suchanek. 2016. Ten Years of Knowledge Harvesting: Lessons and Challenges . In *Data Engineering Bulletin* .