Toward Visual Interactive Exploration of Heterogeneous Graphs
Irène Burger, Ioana Manolescu, Emmanuel Pietriga, Fabian M Suchanek

To cite this version:
Irène Burger, Ioana Manolescu, Emmanuel Pietriga, Fabian M Suchanek. Toward Visual Interactive Exploration of Heterogeneous Graphs. SEData 2020 - Workshop on Searching, Exploring andAnalyzing Heterogeneous Data in conjunction with EDBT/ICDT, Mar 2020, Copenhagen, Denmark. hal-02468778

HAL Id: hal-02468778
https://inria.hal.science/hal-02468778
Submitted on 6 Feb 2020

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.
Toward Visual Interactive Exploration of Heterogeneous Graphs

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

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

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

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 original dataset, e.g., a node for each tuple and attribute from a

Figure 1: Sample screen shot of ConnectionLens GUI [1] showing an answer to a three-keyword query.
We present our analysis of the ways in which ConnectionLens (Section 3). We present some preliminary results, and perspectives on co-occurring in a dataset denote an interesting association, e.g., alternatively, datasets can play a very significant role: (to focus just on connections between users and/or hash tags. Alternatively, datasets may sometimes be transformed into abstract data structures, e.g., if in a social network graph, we want to 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 whole group 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 @Francois_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 root 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
While relationships between them can be built as follows.

Researchers. This decision materializes the concept of a node labeled "Researcher". Given a graph \( G(d) \) such as shown in the figure, corresponding to a dataset \( d \), a graph \( G(d) \) 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, 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 \( Q \) after extracting entities \( p \) from \( d \); the priority is computed as the length of the path from the occurrence to the parent until the dataset root. \( Q \) is a priority queue (\( \tau \) is Extracted Entities, \( \lambda \) is a long text, say, a politician’s speech), no attribute of \( EN(o) \) is extracted from \( p \).

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 \( T = \{ \text{Person}, \text{Location}, \text{Organization} \} \). 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) type.

(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 \( \tau \in T \), we compare \( \lambda \), using Embedded Word Distances [2], to a manually chosen set of keywords \( W_\tau \), and select the type \( \tau_1 \) to which \( \lambda \) is closest. When \( \lambda \) is "Laureate", \( \tau_2 \) is Person. Then:

- **If** \( \tau_2 \neq \tau(o) \), the parent \( p \) comprising \( o \) is "not about \( o \"\); \( p \) likely describes something else. In our example, where \( \tau(o) \) is Person, if \( \tau_2 \) 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 \( \tau_2 = \tau(o) \), we consider that the pair \( p \) 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 \( \tau(c) = \tau_2 \). 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 \( \tau = \text{PERSON} \), then for each edge \( p \rightarrow c \), where \( c \) is the child of \( p \) with type \( \tau_2 \) and \( \alpha \) 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 \( \alpha \) 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 entity 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 inter-connected 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 relations (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 \( o_1, o_2 \) 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.
We implemented in Python the algorithm outlined previously. We worked with ConnectionLens. Preliminary results, that rely on a manual representation of the nodes themselves, as is typically done in multivariate network visualization, are composed of four datasets: two JSON documents describing graphs that would not be amenable to visualization considered in their entirety. Here again, we rely on Graphies [5] to enable 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 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.

We implemented in Python the algorithm outlined previously. We present some quantitative results based on tests made over four heterogeneous datasets denoted $DS_1$, $DS_2$, $DS_3$ and $DS_4$. $DS_1$ is composed of four datasets: two JSON documents describing 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.

We present some quantitative results based on tests made over four heterogeneous datasets denoted $DS_1$, $DS_2$, $DS_3$ and $DS_4$. $DS_1$ is composed of four datasets: two JSON documents describing 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.

We present some quantitative results based on tests made over four heterogeneous datasets denoted $DS_1$, $DS_2$, $DS_3$ and $DS_4$. $DS_1$ is composed of four datasets: two JSON documents describing 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.