A Unified Visual Exploration Framework for (Semi-)structured Data

Théo Bouganim
Inria Saclay
Institut Polytechnique de Paris
Palaiseau, France
theo.bouganim@inria.fr

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

Emmanuel Pietriga
Inria Saclay
Université de Paris-Saclay
Gif-sur-Yvette, France
emmanuel.pietriga@inria.fr

ABSTRACT

Application datasets are shared or published in a variety of technical formats, structured or semi-structured. Visualization tools can help users explore and understand those data, but are typically designed for one data format, *e.g.*, relational, RDF, XML, or even a domain-specific vocabulary such as FOAF for RDF. We present a novel framework that, given as input a dataset of one among many different formats, automatically derives an informative graph structure in the spirit of Property Graphs (PGs), from which an interactive visual interface is created. Our framework leverages prior work on integrating heterogeneous data and abstracting (or summarizing) it. The novelty here lies in interpreting each graph node through this abstraction process, and combining it with an interactive visual representation.

VLDB Workshop Reference Format:
Théo Bouganim, Ioana Manolescu, and Emmanuel Pietriga. A Unified Visual Exploration Framework for (Semi-)structured Data. VLDB 2024 Workshop: BigVis.

1 MOTIVATION

Semi-structured data can be encoded in a variety of formats, including RDF graphs, property graphs, JSON data objects and XML trees. Serialization of such data structures yields textual representations that are easily persisted and transported, facilitating data interchange. Such textual representations can also be loaded and edited in raw text and code editors. But working at this low level of abstraction is often tedious: users experience difficulty understanding, analyzing and interactively manipulating the data, primarily because of the unidimensional nature of this textual representation.

Visual tools and frameworks can support users in their data understanding and analysis tasks. But those tools are typically designed for one particular format, *e.g.*, RDF or XML, or even for one particular domain-specific vocabulary based on those formats, *e.g.*, FOAF or RSS. The goal of this work is to design and develop a framework that can generate interactive visual representations from data in any format (higly structured, i.e., relational, or semi-structured). Based on the basic premise that any kind of data can be modeled as a graph, our framework transforms the imported data into a generic graph structure from which node-link diagram representations can be derived. The research question we focus on is as follows: can we automatically produce, from any input data, a graph structure amenable to visual representation that can effectively support users in data exploration and manipulation tasks?

Our framework primarily uses node-link diagrams for the representation of this structure, as such visualizations are generic and fairly straightforward to understand. But as they suffer from scalability problems – quickly becoming illegible as the graph's size or density increases – we favor an approach based on incremental exploration. Rather than represent the entire graph, the framework lets users identify points of entry into the graph – nodes of particular interest through, *e.g.*, a keyword search or the filtering of a list based on criteria such as node type or facets. A subgraph is then generated from the selected nodes, serving as the basis for the interactive exploration of the whole graph: based on users' interests – *i.e.*, when following selected links – additional nodes are fetched and visualized, and branches of lesser interest can be elided to keep the visual representation manageable.

This paper is organised as follows. Section 2 gives a brief overview of related work in data management systems and interactive graph exploration. Section 3 describes recent data management research results, based on data abstraction, that our approach builds upon. Section 4 describes the new ingredients we contribute to generate property graphs from arbitrary input data based on abstraction. Section 5 studies the performance of our data processing and analysis pipeline. Section 6 then gives an overview of the visualization tool we are currently designing to display those graphs.

2 STATE OF THE ART

We outline the main areas and results in trying to help users cope with large data volumes: graph exploration from a database perspective; and interactive graph visualisation. Compared to these, the novelty of our work is: (*i*) we target structured or semi-structured data from a variety of models; (*ii*) we leverage property graphs as vehicles for interactive visualization.

In this paper, we present our ongoing work toward a first objective: guiding user navigation in an unknown graph, from a given node, to its neighbours. This is but one among many, complementary graph exploration techniques.

2.1 Graph exploration in Data Management Systems

The traditional way of interacting with the data assumes users write queries, which requires a certain level of technical skills, as well

Proceedings of the VLDB Endowment. ISSN 2150-8097.

as some familiarity with the data. To handle situations where one or both are lacking, prior work has investigated ways to (*i*) help users form structured queries, or to (*ii*) search the data via simpler means, such as keyword search and/or flexible – as opposed to structured – querying. Below, we review recent data exploration works targeting graph databases.

Modern graph query languages such as GPML [19] or the JEDI [1] SPARQL extension allow asking for paths between nodes matching some query variables. GPML allows asking for paths between variables even if the users know nothing about the structure of the path; in contrast, SPARQL 1.1 and JEDI require users to combine known graph edge labels in their regular path queries.

In keyword-based search (KBS, in short) [3, 4, 16, 39], one asks, *e.g.*, for connections between "Alice", "Nice" and "IBM". Here, a result is a tree including three nodes whose labels match respectively these keywords. KBS is handy when users know keywords (entities) to search for. It does however have disadvantages: finding all answers may be costly, and it may be hard to "guess" entities that are connected. To remove the latter hurdle, one approach [6] is to automatically enumerate the paths between pairs of connected entities of specific types, e.g., between People and Organizations, that are considered most meaningful.

Other systems let users build SPARQL queries incrementally, *e.g.*, [27] for conjunctive SPARQL, and [25] for queries with aggregation. Many graph exploration methods exist, see, *e.g.*, [26], in particular coming from the data visualisation community; we discuss them below.

2.2 Interactive graph exploration

The literature on interactive graph visualisation and exploration is extremely rich and varied [38]. We focus our overview of this research area on works that deal specifically with knowledge graphs and some types of linked data browsers [17], as opposed to multivariate networks at large (covered in a recent survey [28]).

Some of the earliest visualisation tools represented raw graphs as node-link diagrams [29]. As these representations were quite verbose and generic, their scalability was limited. Various approaches were devised to address this issue. For instance, a language inspired by Cascading Stylesheets (CSS) was designed to customize the appearance of nodes and links [30]. The representation remained quite verbose, however. Later systems displayed the graph as a node-link diagram but at a higher level of abstraction, aiming to support specific exploration tasks. LodLive [13] and Visor [32] display the graph in a dynamic manner. The graph is not shown in its entirety: starting from a subgraph, additional nodes and links get displayed on-demand, aggregating part of the structure and thus limiting visual clutter. RelFinder [23], as the name suggests, shows paths connecting pairs of resources, starting with the shortest ones. Visor can also represent paths connecting elements several hops away in the graph structure, taking edge direction into account.

Other knowledge graph exploration interfaces depart from the node-link diagram representation, and rather focus on the visualisation of *sets* of resources. This can be with faceted browsing [22, 24, 37], by aggregating properties based on path characteristics for the properties of interest [18] to produce meaningful views, or through the visualisation of specific characteristics of

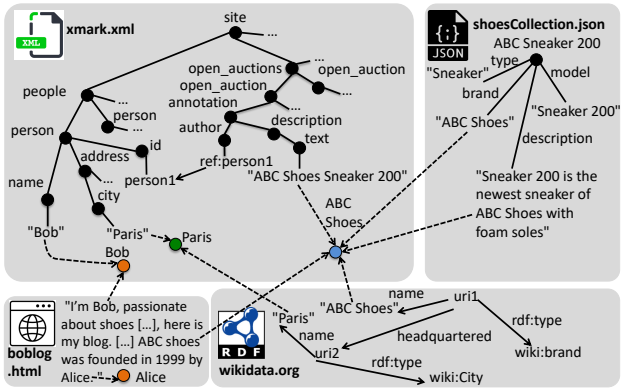

Figure 1: Sample ConnectionLens graph built from four datasets.

those resources' properties, such as the distribution of values for a given property [11, 12, 36]. Others focus on RDF visualisation allowing the user to access more details [14, 31]

In this context, we present our ongoing work toward a first objective: guiding user navigation in an unknown graph, from a given node, to its neighbours.

3 PRELIMINARIES

We first recall concepts and results our work builds upon.

3.1 ConnectionLens and normalized graphs

ConnectionLens [2, 3, 16] is a system capable of integrating heterogeneous datasets into a graph, as illustrated in Figure 1. We integrate four datasets (corresponding to the gray areas) in an e-commerce scenario. In this example, we use: (*i*) an XMark [35] XML auction document where ABC shoes were sold; (*ii*) a JSON document detailing different shoe models; (*iii*) an RDF fragment from Wikidata, about the shoe maker, and (*iv*) an auction user's HTML blog page. ConnectionLens enriches the graph by extracting Named Entities (NEs), e.g., person, organisations or locations, each added as a new node to the graph; we use one color for each NE type.

As Figure 1 shows, some edges are labeled and some edges are not, due to the structure of the original data. For instance, all edges in an RDF graph are labeled; property graphs (PGs) are ingested by turning each node (or edge) property into a standalone node, connected to its parent node (edge) via an edge labeled with the property name. In contrast, the natural modeling of XML documents leads to unlabeled edges between parent and child elements, while edges between elements and attributes are labeled; most edges in JSON documents are not labeled, *etc.* To facilitate the analysis of heterogeneous data graphs, ConnectionLens graphs are *normalized*: each labeled edge is replaced by a new intermediate node carrying the original edge's label, as well as two unlabeled edges connecting it to the two ends of the original edge.

Notations. We introduce the following notations. A normalised directed graph G is a triple of the form $\langle \mathcal{V}, \mathcal{E}, \mathcal{L} \rangle$, where: $\mathcal{V}$ is a set of nodes; $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of (unlabelled) directed edges; and $\mathcal{L}$

is the set of node labels. Further, we write $(s.n_{i_1}...n_{i_p}.t)$ to denote a directed path from source node s to target node t, of length p. Finally, $\mathcal{P}$ designates the set of all directed acyclic paths in G.

3.2 Collections

Abstra [7, 8] is a *data abstraction* system developed recently. Given a relational, JSON, XML, property graph (PG) or RDF dataset, from the ConnectionLens graph corresponding to the dataset, Abstra automatically computes a description meant to give humans a *first-sight overview* of a dataset. This description is a flavor of Entity-Relationship (ER) diagram, of which many variants have been proposed in the literature. In particular, when modeling relational databases, an entity's attributes only have atomic values [33]; in contrast, the value of an attribute of an Abstra entity can have a nested structure. For instance, the value of an "address" attribute can be a set of two locations, each specified as street, city, and country; one location may also have an extra "getting there" field structured in successive steps, *etc.*[1]

As an intermediary step towards abstracting datasets, Abstra groups nodes in **equivalence classes**. Abstra's notion of equivalence [8] seeks to capture the judgment of users who authored the data; it can be seen as reverse-engineering the data design guidelines associated to each data model, and which dictate how to describe application data in that model. For instance, human users immediately understand that "two articles are similar (conceptually equivalent)" as soon as they both have, *e.g.*, a title and some authors. In order to work with graphs generated from a variety of data sources, we leverage Abstra's node equivalence, specified in [8]. A group of nodes considered equivalent by Abstra are said to be part of a **collection**.

Each collection has a label. For XML, PGs or JSON, the label of the collection is the label shared by the nodes of the collection, while for RDF, this label is the most specific type shared with the other nodes of the collection.

3.3 Entities, boundaries and relationships

Having grouped all nodes in collections, Abstra performs the following steps:

(1) Organize the collections in a directed graph, having an edge $C_1 \rightarrow C_2$ whenever in the original graph, a node of collection C_1 has an outgoing edge toward a node of collection C_2; we call this the **collection graph**. Figure 2 shows the collection graph obtained from the XML data at the top left in Figure 1: each collection is shown as a box, labeled with an XML element name, e.g., initial, or with a label such as #initial, which denotes the set of text nodes that are children of the initial elements. The colors of the collection boxes will be explained shortly.

(2) Choose a set of collections, each of which is designated as the root of an *entity, i.e.*, a core set of things described by the dataset. Each entity collection e is also attached a set of *attributes, i.e.*, other collections accessible from e, and seen as describing the e entities. Abstra defines the *boundary* of an entity e as the set of these collections describing e, together with edges between them. For example, in Figure 2, the entity root collections are orange-filled, while their boundaries are delimited by the light orange shapes.

(3) Based on the chosen entities, Abstra turns any directed path from an entity collection e_1 to another entity collection e_2, into a *relationship* between the entities. In Figure 2, the annotation and author collections (blue background) materialize the relation between the open auctions and the people.

[1]Sample abstractions computed by Abstra can be seen online at https://team.inria.fr/cedar/projects/abstra/.

4 UNDERSTANDING DATASETS THROUGH THE PRISM OF ABSTRACTION

Abstraction leads to a compact E-R representation of a dataset, where users can see at a glance the main entities and relationships. However, abstractions are not interactive, and they do not enable finding information about any individual node. We seek to leverage abstractions, to help users interactively explore the original data graph at the node level. This raises two challenges, which can be seen as *losses of information (or precision)* ; we detail them in Section 4.1. Then, we describe our approach for overcoming these, and reaching our goal.

4.1 Abstraction-induced information loss

A first observation is that abstractions may *hide (not reflect) some nodes from the original graph.* This happens in several cases.

- To keep the E-R diagram of manageable size, Abstra users may specify an upper bound on the number of entities. Thus, node collections that are relatively rare, in a complex-structure graph, may not be reflected in the Abstra output at all: not as entities, not even in other entities' boundaries.
- For visual simplicity, Abstra relationships are *simple paths* from an entity to another, of the form $e_1 \rightarrow k_1 \rightarrow k_2 \ldots \rightarrow e_2$ for some collections k_i. If a collection k_j is reachable only from one of the k_i's in a relation, k_j is visible neither in the relation, nor an attribute within the boundary of e_1 or e_2. For example, in Figure 2, the description and text collections under annotation would not be visible in the abstraction.
- Finally, Abstra rejects 1-node equivalence classes as entity roots, e.g., the site, open_auctions and people singleton collections in Figure 2. Such a singleton can be seen as only a "container" for a larger collection. If such singletons are not included in an entity's boundary, they may not be visible in the abstraction.

Second, *summarization may consider equivalent nodes with slightly different structure, e.g.*, two person p_1, p_2 having name and address, even if only p_1 has an email address. Such simplifications are common when summarizing graphs. Further, for efficiency, Abstra selects entity collections, and their boundaries, *directly on the collection graph*, not on the original graph. Thus, equivalent nodes, indistinguishable in the abstraction, may have different internal (surrounding) structures, which should be faithfully reflected for each, when exploring the data at the node level.

Intuitively: *abstraction paints with wide brushstrokes; node-oriented exploration requires a fine brush.* This is why we need

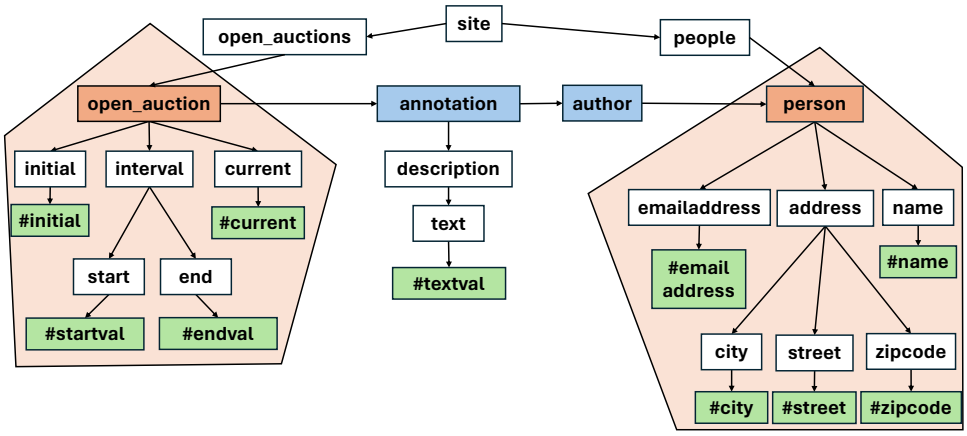

Figure 2: Collection graph obtained from the XML document in Figure 1.

to extend the node interpretation (to which entities or relationships do they belong? what part do they play there?) to *each* node individually. We discuss this in Sections 4.2 and 4.3.

4.2 A role for every node

We start by dividing all the nodes into four basic *roles*, as illustrated in Figure 3, using one node background color for each role (differently from Figure 2, Figure 3 represents *individual nodes and edges*):

- *Value nodes* (green fill), denoted $\mathcal{V}_V$, are the leaves of the graph. They have no children, but contain the actual data values.
- *Entity root* nodes (orange), denoted $\mathcal{V}_E$. These are nodes identified by Abstra as entry points, or top nodes, in its entities. Each Abstra entity, stripped of its relations, can be seen as a DAG, containing nodes that describe the entity root. Thus, $\mathcal{V}_E$ nodes are exactly those from entity root collections.
- *In-relationship* nodes (blue), denoted $\mathcal{V}_R$. These are data nodes that are, in the data graph, along a directed path from an entity root node (previous role in this list) to another. For example, in Figure 3, there are exactly two such nodes.
- *Structural nodes* (white), denoted $\mathcal{V}_S$, are all other nodes in the graph.

We compute $\mathcal{V}_E$ as: the nodes assigned by Abstra to a collection that is an entity root. We compute $\mathcal{V}_V$, $\mathcal{V}_R$, and $\mathcal{V}_S$ directly on the graph and also based on $\mathcal{V}_E$.

4.3 Boundaries to which a node belongs

Our next task is to assign each $\mathcal{V}_V$, $\mathcal{V}_R$ and $\mathcal{V}_S$ node into the boundaries of one or more entity instances. We do this by traversing the graph starting from each $\mathcal{V}_V$ node, going backwards, along all acyclic paths, until we reach an entity root, i.e., a $\mathcal{V}_E$ node. All the nodes (and edges) traversed on an acyclic path that reaches a node $n_E \in \mathcal{V}_E$, are part of the boundary of n_E. In a similar fashion, we also compute a boundary for each in-relationship node, except that in this process, we stop when reaching a node from $\mathcal{V}_R$, and we attach boundaries to $\mathcal{V}_R$ nodes. Intuitively, such boundaries capture "all attributes needed to describe relationships between nodes". Entities are described by possibly nested attributes, and the nodes along the relationship paths may also have their own attributes. We say such attributes are part of the *boundary* of the respective nodes (either entity root, or relationship node). Visually, in Figure 3, boundaries are represented as shaded pink areas. As illustrated in the Figure, boundaries can overlap: a city is shared by two people living there. In general, boundaries overlap when several entity or relationship instances are described by the same nodes (or subgraphs).

4.4 Deriving Property Graphs

Property Graphs (PGs, in short) are currently very popular for representing (semi)structured data. PG schemas can be derived from dataset abstractions [20]; in our context, we need to derive PGs from normalized graphs, using the abstractions. A PG consists of a set of nodes, and possibly a set of directed edges. Each node and edge can have: one or more labels; and a set of attributes, which have a name and a value; values are atomic. A PG node or edge may have one or more labels, which serve to distinguish different kinds of nodes. From a data management perspective, PGs are preferable to simple graphs, such as those built by ConnectionLens (Figure 1), or RDF graphs, because if nodes are stored with their attributes, less joins (or navigation) are needed to retrieve a node's content, and similarly for edges. From a user's perspective, also, the encapsulation of attributes in PG nodes and edges makes them intuitive and easy to grasp.

We turn each entity instance rooted in a node $e \in \mathcal{V}_E$ into a PG node pg_e, as follows.

- Node pg_e has an internal identifier attribute id, whose value is the one created for it by ConnectionLens.
- Recall that e may have multiple values for a child label l. Since this is not allowed in PGs, we attach to pg_e one attribute whose name is l, and whose value we compute by serializing all the values of l within a JSON list of the form $[lv_1, lv_2, \ldots, lv_k]$.
- Further, e's attributes may have nested values, which PGs do not model. Thus, a non-atomic value of an attribute

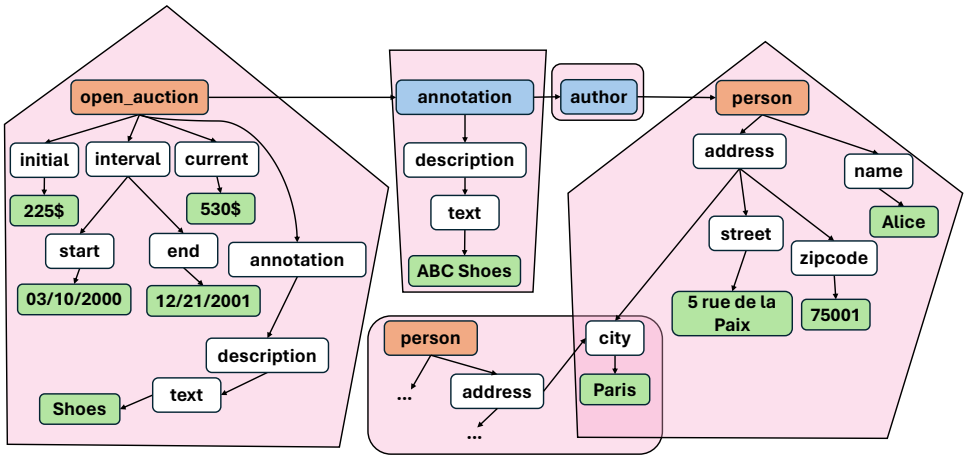

Figure 3: Boundaries of entity instances (in the data graph).

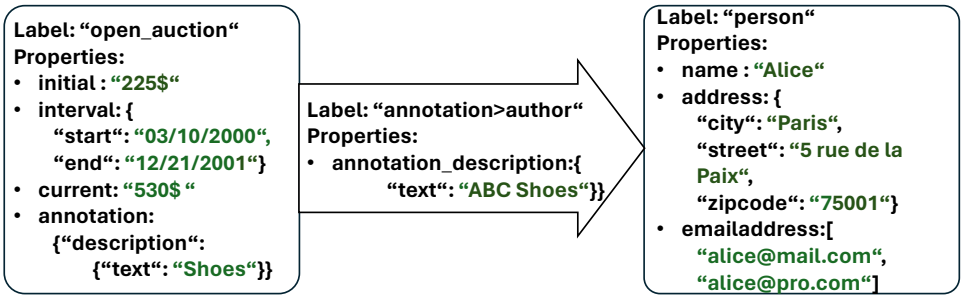

Figure 4: Property graph obtained from the entity and relationship instances in Figure 3.

labeled l is serialized into a JSON snippet of the form l: $\{la_1 : \ldots, la_2 : \ldots, \ldots, la_p : \ldots\}$, where la_i's are the nested children of l. JSON lists and maps, nested within each other as needed, are used to model attribute values that are multiple, respectively, non-atomic (have nested children).

- Given the semantics PG labels have, the label(s) of pg_e should reflect its category/type. Thus, pg_e, and all the PG nodes derived from nodes in the same (entity root) collection as pg_e, take their labels from that collection: if they are XML elements, the element name; if they are JSON nodes, the label path from the JSON root to the node; if they are typed RDF node, one label for each type; for relational datasets, the labels derive directly from the schema, etc.

A graph relationship r of the form $e_1.n_1.\ldots.n_l.e_2$ where $e_1, e_2 \in \mathcal{V}_E$ and $n_i \in \mathcal{V}_R$ is transformed into a (single) PG edge from the node pg_1 created from e_1 as above, to the node pg_2 corresponding to e_2. This edge is labeled with the concatenation of the labels of all the collections along the relationship. For each node along the relationship, we compute the properties in the same way they are computed for PG nodes. Finally, the PG edge accumulates the properties of each node along the relationship, each property carrying the label of the node it comes from, as a prefix of its own label. This way, all information is kept during the transformation. In fact, if we have two PG nodes person, `Alice` and `Bob` connected by a relation in the original graph of two nodes: `package expedition` and `package reception` both having a `date` attribute, the resulting PG edge would have two attributes `date` that we need to distinguish.

For illustration, Figure 4 shows two PG nodes and an edge resulting from the data in Figure 2. We have added two emails to the node at the right (not present in the data previously depicted), to illustrate the handling of multi-valued attributes.

The PG thus obtained reflects all the nodes in the Abstra-chosen entities and relationships, as well as all their boundaries. Running Abstra with no upper limit on the number of selected entities maximizes the part of the original dataset we transform into a Property Graph.

5 EXPERIMENTS

When users import data into the framework, we want it to be available for visualisation as quickly as possible. We thus evaluate the performance of our method as the time taken by each data processing step, and the scalability in the data size. After describing our datasets (Section 5.1) and experimental settings (Section 5.2), we discuss our experiments and their results.

name	size (bytes)	nb. of nodes	nb. of edges
XMark025	29.220.287	945.366	1.062.437
XMark05	58.005.779	1.876.628	2.122.577
XMark1	116.517.344	3.733.934	4.246.835
BSBM3K	208.986.518	971.755	1.458.780
BSBM4K	313.589.776	1.453.549	2.189.198
BSBM6K	511.100.393	2.360.702	3.563.778

Table 1: Datasets and metrics on the generated graphs.

5.1 Datasets

We used two different controllable-size datasets for our experiments. XMark [35] is an XML benchmark describing auctions, products and buyers. We generated files with scale factors 0.25, 0.5, and 1, respectively. We fed them to Abstra, leading to normalized graphs having from 1M to 4.24M edges. BSBM [9] is an RDF benchmark describing an e-commerce scenario, comprising products, vendors, and consumers posting reviews about products. Again, we generated three datasets of increasing size (Table 1), leading to normalized graphs of .67M to 2.18M edges. The numbers of nodes and edges in the normalized Abstra graphs are also reported in Table 1.

5.2 Settings

Our computation of roles, boundaries, and conversion to property graphs is implemented in Python 3.6. First, we call Abstra [8] to build the normalized graphs and collections, which we read from Postgres. We then measure the time taken by our algorithms to convert the data to a PG, also persisted in Postgres. We used a MacBook with Apple M1 Pro chip with 16 GB of memory.

5.3 Results

Figure 5 and 6 shows the time taken by each operation in our framework, as a function of the number of edges in the normalized graph. We note that each operation time grows linearly with the graph size (number of edges). The longest times are to compute the boundaries (Section 4.3) and write the PG in the database. The times are shorter for BSBM (RDF), because its relations and boundaries are less deep, simplifying our computations. Finally, the total time is acceptable for the processing of large datasets and as it is linear, we cannot expect a lot of improvement. We could though parallelize operations.

6 DATA GRAPH EXPLORATION

The property graphs generated by the above data transformation process are all structured in the same manner, amenable to a generic form of exploratory visualization.

We opt for a visualization based on node-link diagrams as such diagrams are conceptually simple to understand. But, as node-link diagrams scale poorly with the size and density of graphs, we employ strategies to keep the visual representation compact. Figure 7 illustrates part of the interface, displaying a summarized view of an XMark document, detailing a *Person*'s properties.

Our visualizations are implemented using D3 [10] and integrated in the Connection Studio platform [5].

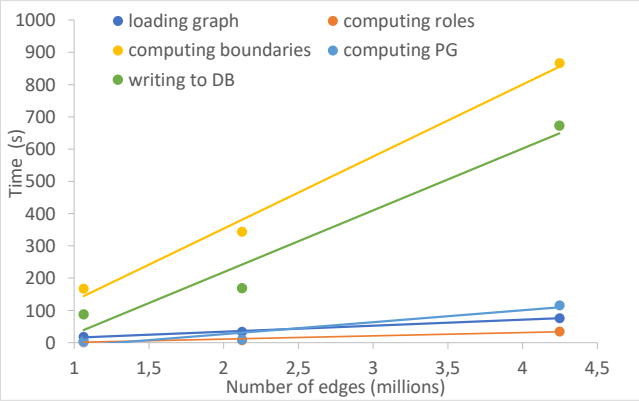

Figure 5: Data processing times on XML datasets.

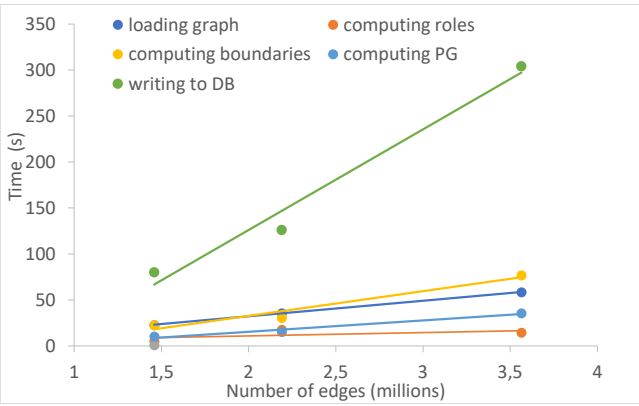

Figure 6: Data processing times on RDF datasets.

6.1 Initial query

For many datasets, showing the whole graph results in a visualization that bears little value because of the large number of nodes and links. We rather opt for an initially blank workspace that users will populate incrementally with entities and relationships of certain types, as in Graphies [34].

This obviously requires enabling users to select points of entry in the graph. This can be a random object if the user has no clear idea of where to start and merely wants to discover the data by browsing it.

If the user has particular questions in mind, it is also possible to specify a node-set to start from. Such a set can be obtained from a keyword search, in which case the workspace will get populated with objects (entities and relations) matching the query. Since values are nested in JSON structures in PG objects, to optimize searchability, the search is operated on the normalized graph. We store mappings between these nodes and the PG nodes (or relationships) they are part of.

Users can specify if they are looking for a value, an entity root, a relation, or a property containing the keyword. The returned objects are entities and/or relations that contain the keyword(s). Users can also search for PG nodes through ad-hoc property search

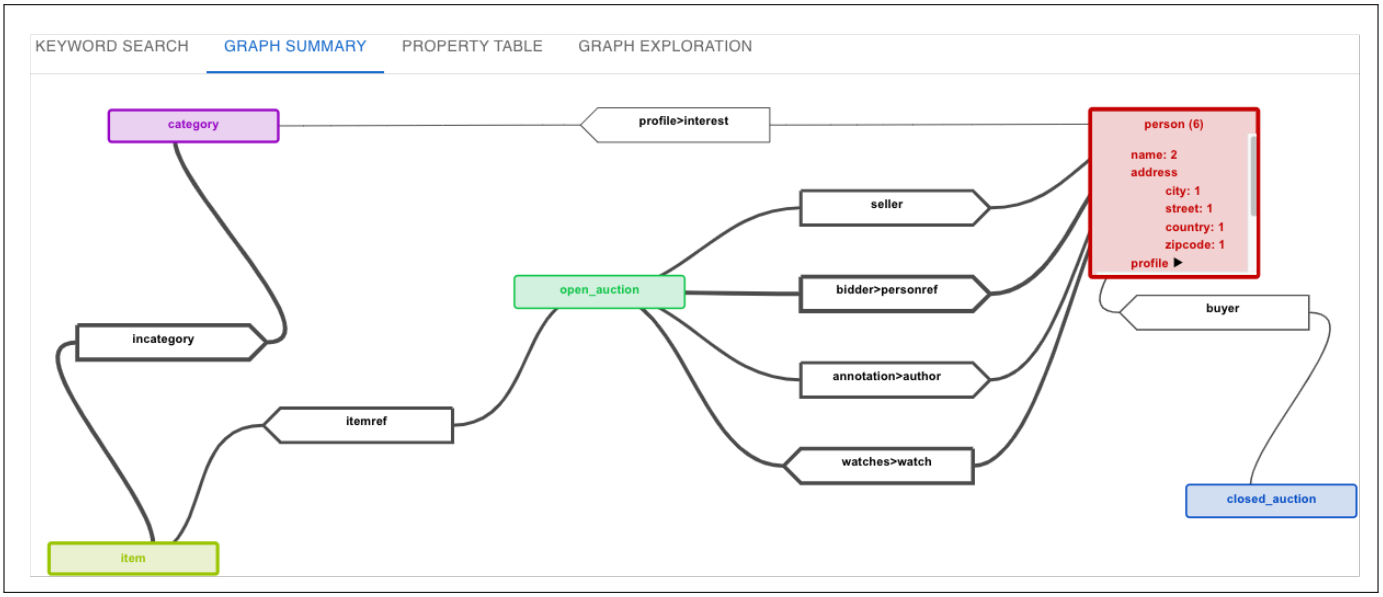

Figure 7: Property graph summary for an XMark document with a person's properties expanded.

forms, e.g., find the Person named Alice, or all Person nodes living in NY.

Another option consists in generating a visual summary of the graph that reflects the abstractions computed by Abstra (Section 4). As illustrated in Figure 7, this visualization shows a summary of the structure of the property graph, representing each collection of entities and the collection of relations in the property graph and how they are connected.

This summary can help users identify entity or relation types of interest that can then be selected and serve as an entry point into the actual property graph. Selecting an entity or relation takes users to a property table that lists all instances of the corresponding collection, together with their properties arranged in columns. The header of each column is the path from the root to the property. Selecting one particular item in this list takes users to the *Graph Exploration* panel, centering the viewport on the corresponding element.

6.2 Compactness

We described how we compute a property graph from imported datasets in Section 4. The nodes and edges of this graph are rich objects holding multiple properties that we represent nested inside the corresponding geometrical primitives: boxes that can be expanded and whose content can be scrolled for nodes; and curves decorated with an arrow shape for links. This arrow shape conveys the relation type and its direction. It can be expanded and scrolled as well, to show properties associated with the corresponding type of relationship. We favor nesting of key-value pairs in both cases for the sake of compactness.

Nodes are laid out using D3's force simulation (package d3-force) [10], setting a link force that leaves reasonable empty space around shapes to expand them. Boxes decorating links are treated as nodes in the simulation driving the layout, link curves being composed of two Bézier curves, one on each side. Clicking on a shape expands that shape to reveal the nested list of properties. That list is made scrollable if it overflows the parent node shape, as illustrated in Figure 7 with the *person* collection.

Each node collection is assigned a different color hue, consistent across views (actual property graph and graph summary). In the *Graph Summary* view, stroke width (nodes and links) is mapped to the group's cardinality, giving a rough idea about its number of instances.

6.3 Incremental exploration

The *Graph Summary* view is typically limited to a few dozen boxes with corresponding links as it aggregates entities by collection or relationship type. The *Graph Exploration* view, however, shows the actual instances themselves and can quickly become cluttered. Users start from a blank workspace that they will populate with a small initial node-set, adding and pruning branches incrementally as they follow relations of interest.

By default, only a subset of a node's relations is shown, chosen based on their relevance. Selecting one of those relations will fetch and display the corresponding target node, which can then become the primary focus of attention. Providing users with a seamless and efficient way to navigate the graph is key, avoiding too many explicit actions to manually expand branches and prune links of little interest while keeping relevant branches readily accessible at the same time. This interaction design phase is currently on-going, as discussed next.

7 ONGOING AND FUTURE WORK

On the data transformation side, our pipeline can be improved using parallelization to increase performance. The pipeline could also be made into a universal (semi)-structured data converter to Property

Graphs by allowing other PG schemas than the one computed with Abstra, letting users choose entity root collections.

On the user interface side, we are currently exploring several areas for improvement. As mentioned earlier, nodes can have many properties and many relations. We expose those properties through a scrollable list, and are investigating different strategies to sort them, such as computing a score of pertinence for each object, inspired by KL-divergence with pseudo-relevance feedback [27] or personalized PageRank [15], showing to the user the top-k ranked objects. We can also allow the user to choose the entity roots collection, thus, allowing other PG schemas than the one computed from Abstra. Such transformation pipeline would be a universal (semi)-structured data converter to Property Graphs.

Other enhancements include: using word scale visualizations [21] to show the value distribution per property across all nodes of a certain collection; using input modalities beyond keyboard and mouse to interact with elements of the node-link diagram, such as pen and touch [34]. The latter is part of a larger interaction design effort to ease navigation in the node-link diagram, streamlining navigation (interactive traversal) and graph element visibility management actions, thanks to the flexibility and additional expressive power of those two modalities combined.

We will be conducting a user-study to evaluate the quality of the data transformation and of the exploration and visualization tool once it is ready.

Beyond node and link visibility management, we are investigating different input modalities beyond mouse and keyboard to interact with the graph view, such as pen and touch. We believe that these have much potential because of the flexible yet precise selections they afford in a direct manipulation context.

ACKNOWLEDGMENTS

This work is partially funded by the AI Chair SourcesSay project (ANR-20-CHIA-0015-01).

REFERENCES

[1] Christian Aebeloe, Vinay Setty, Gabriela Montoya, and Katja Hose. 2018. Top-K Diversification for Path Queries in Knowledge Graphs. In *ISWC Workshops*.

[2] Angelos-Christos Anadiotis, Oana Balalau, Théo Bouganim, Helena Galhardas, Mhd Yamen Haddad, Ioana Manolescu, Tayeb Merabti, and Jingmao You. 2021. Empowering Investigative Journalism with Graph-based Heterogeneous Data Management. *IEEE Data Engineering Bulletin* (2021). https://arxiv.org/abs/2102.04141

[3] Angelos Christos Anadiotis, Oana Balalau, Catarina Conceiçao, Helena Galhardas, Mhd Yamen Haddad, Ioana Manolescu, Tayeb Merabti, and Jingmao You. 2021. Graph integration of structured, semistructured and unstructured data for data journalism. *Inf. Sys.* (2021). https://doi.org/10.1016/j.is.2021.101846

[4] Angelos Christos Anadiotis, Ioana Manolescu, and Madhulika Mohanty. 2023. Integrating Connection Search in Graph Queries. In *ICDE*.

[5] Nelly Barret, Simon Ebel, Théo Galizzi, Ioana Manolescu, and Madhulika Mohanty. 2023. User-friendly exploration of highly heterogeneous data lakes. In *CoopIS*.

[6] Nelly Barret, Antoine Gauquier, Jia-Jean Law, and Ioana Manolescu. 2023. PathWays: entity-focused exploration of heterogeneous data graphs. In *ESWC*.

[7] Nelly Barret, Ioana Manolescu, and Prajna Upadhyay. 2022. Abstra: Toward Generic Abstractions for Data of Any Model. In *CIKM*. https://doi.org/10.1145/3511808.3557179

[8] Nelly Barret, Ioana Manolescu, and Prajna Upadhyay. 2024. Computing Generic Abstractions from Application Datasets. In *EDBT*.

[9] Christian Bizer and Andreas Schultz. 2009. The Berlin SPARQL benchmark. *IJSWIS* 5, 2 (2009).

[10] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 Data-Driven Documents. *IEEE TVCG* 17, 12 (2011). https://doi.org/10.1109/TVCG.2011.185

[11] Adrian Brașoveanu, Martab Sabou, Arnoa Scharl, Alexandera Hubmann-Haidvogel, and Daniela Fischl. 2016. Visualizing statistical linked knowledge for decision support. *Semantic Web Journal* 8, 1 (2016). https://doi.org/10.3233/SW-160225

[12] Josep Maria Brunetti, Sören Auer, Roberto García, Jakub Klímek, and Martin Nečaský. 2013. Formal Linked Data Visualization Model. In *IIWAS* (Vienna, Austria).

[13] Diego Valerio Camarda, Silvia Mazzini, and Alessandro Antonuccio. 2012. LodLive, Exploring the Web of Data. In *I-SEMANTICS* (Graz, Austria). http://doi.acm.org/10.1145/2362499.2362532

[14] D. V. Camarda, S. Mazzini, and A. Antonuccio. 2012. LodLive, exploring the web of data. In *Proceedings of the 8th International Conference on Semantic Systems*.

[15] Soumen Chakrabarti. 2007. Dynamic personalized pagerank in entity-relation graphs. In *Proceedings of the 16th international conference on World Wide Web*.

[16] Camille Chanial, Rédouane Dziri, Helena Galhardas, Julien Leblay, Minh-Huong Le Nguyen, and Ioana Manolescu. 2018. ConnectionLens: Finding Connections Across Heterogeneous Data Sources (Demonstration). *PVLDB* 11, 12 (2018).

[17] Aba-Sah Dadzie and Emmanuel Pietriga. 2017. Visualisation of Linked Data - Reprise. *Open Journal Of Semantic Web* 8, 1 (2017). https://doi.org/10.3233/SW-160249

[18] Marie Destandau, Caroline Appert, and Emmanuel Pietriga. 2021. S-Paths: Set-based visual exploration of linked data driven by semantic paths. *Open J. Sem. Web* 12, 1 (2021). https://doi.org/10.3233/SW-200383

[19] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, et al. 2022. Graph Pattern Matching in GQL and SQL/PGQ. In *SIGMOD*.

[20] T. Enache, N. Barret, I. Manolescu, and M. Mohanty. 2024. Finding the PG schema of any (semi)structured dataset: a tale of graphs and abstraction. *SEAGraph workshop* (2024).

[21] Pascal Goffin, Wesley Willett, Jean-Daniel Fekete, and Petra Isenberg. 2014. Exploring the Placement and Design of Word-Scale Visualizations. *IEEE TVCG* 20, 12 (2014). https://doi.org/10.1109/TVCG.2014.2346435

[22] Andreas Harth. 2010. VisiNav: A system for visual search and navigation on web data. *Journal of Web Semantics* 8, 4 (2010).

[23] Philipp Heim, Sebastian Hellmann, Jens Lehmann, Steffen Lohmann, and Timo Stegemann. 2009. RelFinder: Revealing Relationships in RDF Knowledge Bases". In *Sem. Multimedia*.

[24] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. 2006. /facet: A Browser for Heterogeneous Semantic Web Repositories. In *ISWC*. https://doi.org/10.1007/11926078

[25] Matteo Lissandrini, Katja Hose, and Torben Bach Pedersen. 2023. Example-Driven Exploratory Analytics over Knowledge Graphs. In *EDBT*.

[26] Matteo Lissandrini, Davide Mottin, Katja Hose, and Torben Bach Pedersen. 2022. Knowledge Graph Exploration Systems: are we lost?. In *CIDR*.

[27] Matteo Lissandrini, Davide Mottin, Themis Palpanas, and Yannis Velegrakis. 2020. Graph-query suggestions for knowledge graph exploration. In *The Web Conference*.

[28] C. Nobre, M. Meyer, M. Streit, and A. Lex. 2019. The State of the Art in Visualizing Multivariate Networks. *CGF* 38, 3 (2019). https://doi.org/10.1111/cgf.13728

[29] Emmanuel Pietriga. 2002. IsaViz: a Visual Environment for Browsing and Authoring RDF Models. In *WWW Developers Day*. http://www.w3.org/2001/11/IsaViz/

[30] Emmanuel Pietriga. 2006. Semantic Web Data Visualization with Graph Style Sheets. In *SoftVis* (Brighton, United Kingdom).

[31] I.O. Popov, M.C. Schraefel, W. Hall, and N. Shadbolt. 2011. Connecting the Dots: A Multi-pivot Approach to Data Exploration. In *ISWC*.

[32] Igor O. Popov, M. C. Schraefel, Wendy Hall, and Nigel Shadbolt. 2011. Connecting the Dots: A Multi-pivot Approach to Data Exploration. In *ISWC*.

[33] Raghu Ramakhrishnan and Johannes Gehrke. 2003. *Database Management Systems (3rd edition)*. McGraw-Hill.

[34] Hugo Romat, Caroline Appert, and Emmanuel Pietriga. 2021. Expressive Authoring of Node-Link Diagrams with Graphies. *IEEE TVCG* 27, 4 (2021). https://doi.org/10.1109/TVCG.2019.2950932

[35] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J Carey, Ioana Manolescu, and Ralph Busse. 2002. XMark: A benchmark for XML data management. In *VLDB*.

[36] K. Thellmann, M. Galkin, F. Orlandi, and S. Auer. 2015. LinkDaViz – Automatic Binding of Linked Data to Visualizations. In *ISWC*. https://doi.org/10.1007/978-3-319-25007-6

[37] Y. Tzitzikas, N. Manolis, and P. Papadakos. 2017. Faceted Exploration of RDF/S Datasets: A Survey. *Journal of Intelligent Information Systems* 48, 2 (2017).

[38] Tatiana Von Landesberger, Arjan Kuijper, Tobias Schreck, Jörn Kohlhammer, Jarke J. van Wijk, J-D. Fekete, and Dieter W. Fellner. 2011. Visual analysis of large graphs: state-of-the-art and future research challenges. 30, 6 (2011).

[39] Jianye Yang, Wu Yao, and Wenjie Zhang. 2021. Keyword Search on Large Graphs: A Survey. *Data Sci. Eng.* 6, 2 (2021).