Finding meaningful paths in heterogeneous graphs with PathWays

Nelly Barret, Antoine Gauquier, Jia-Jean Law, Ioana Manolescu

Finding meaningful paths in heterogeneous graphs with PathWays

Nelly Barret[a,*], Antoine Gauquier[b,1], Jia-Jean Law[c,2], Ioana Manolescu[a]

[a]*Inria & Institut Polytechnique de Paris, 1 rue Estienne Honoré d'Orves, Palaiseau, 91120, France*
[b]*DI ENS, ENS, CNRS, PSL University & Inria, 45 rue d'Ulm, Paris, 75005, France*
[c]*Columbia University, 116 and Broadway, New York, 10027, USA*

Abstract

Graphs, and notably RDF graphs, are a prominent way of sharing data. As data usage democratizes, users need help figuring out the useful content of a graph dataset. In particular, journalists with whom we collaborate are interested in identifying, in a graph, the *connections between entities*, e.g., people, organizations, emails, etc.

We present a novel method for exploring data graphs through *their data paths connecting Named Entities* (NEs, in short); each data path leads to a tabular-looking set of results. NEs are extracted from the data through dedicated Information Extraction modules. Our method builds upon the pre-existing ConnectionLens platform and follow-up work in the Abstra project, which builds simple, visual ER-style summaries of semi-structured data. The contribution of the present work, and its novelty, is twofold. First, we propose a novel analysis of entity-to-entity paths contained in datasets of any nature, and propose a new method for ranking paths, leveraging a novel Information Extraction (IE) module we built on top of ChatGPT. Second, we present an efficient approach to enumerate and compute NE paths, based on an algorithm which automatically recommends sub-paths to materialize, and rewrites the path queries using these subpaths. Our experiments demonstrate the interest of NE paths and the efficiency of our method for

*Corresponding author

Email addresses: nelly.barret@inria.fr (Nelly Barret), antoine.gauquier@ens.fr (Antoine Gauquier), law.jia.jean@gmail.com (Jia-Jean Law), ioana.manolescu@inria.fr (Ioana Manolescu)

[1]Contributed to the work while affiliated to Institut Mines Télécom and Inria

[2]Contributed to the work while affiliated to Ecole Polytechnique and Inria

computing and ranking them.

Keywords: Data graphs, Graph exploration, Information Extraction

---

1. Motivation and problem statement

The increased digitization of modern life has led to many datasets being produced by automated systems, individual users, and organizations, private or public, companies or governance organizations, NGOs, etc. The W3C recommends using the Resource Description Format (RDF, in short) for sharing data. However, in practice, datasets are shared in many formats: relational or tabular (CSV files, spreadsheets, relational database dumps, etc.); semi-structured documents (HTML, XML, JSON); graphs, either RDF, or Property Graphs (PGs), etc. The diversity of formats is particularly challenging to non-IT users, which need help understanding the dataset structure and what kind of information it could bring them. In recent years, we have carried research together with investigative journalists, with the goal of helping them understand and analyze datasets that they may get access to, and which are often of varied data models.

The ConnectionLens system Anadiotis et al. (2021, 2022) has been developed towards this goal. ConnectionLens turns any (set of) datasets into a single graph, having: (*i*) an internal node for each structural element of the original dataset, e.g., relational tuple, XML element or attribute, JSON map or array, URI in an RDF graph; (*ii*) a leaf node for each value in a dataset, e.g., attribute value in a relational tuple, text node or attribute value in XML, atomic (leaf) value in a JSON document, literal in RDF, or value of a PG node or edge property. There is an edge in the graph for each connection between the data items in the original dataset, e.g., parent-child relationships between XML or JSON nodes, edges connecting each relational tuple node with their attributes, etc. In a relational database, primary/foreign keys may lead to more edges, e.g., the node representing an Employee tuple "points to" the Company tuple representing their employer. This graph view of a relational dataset has been introduced to support keyword-based search in relational databases Agrawal et al. (2002), then extended to (semi)structured documents Hristidis et al. (2003) and used in many follow-up works, see, e.g., Yang et al. (2021); Manolescu and Mohanty (2023).

For instance, the graph in Fig. 1 illustrates a **data journalism scenario** on which we applied ConnectionLens, working with a journalist that sought to identify conflicts of interest in the biomedical domain Anadiotis et al. (2021). Authors

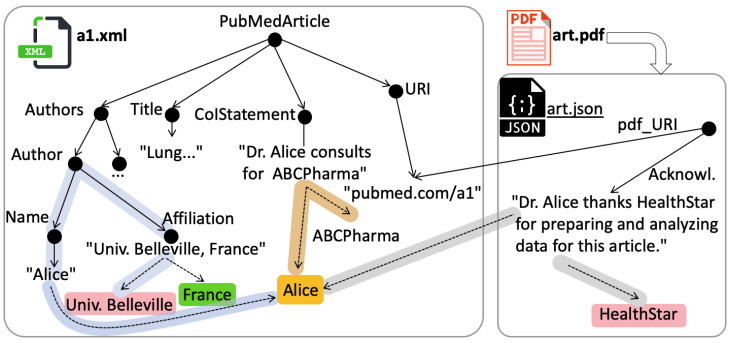

Figure 1: Sample data graph with extracted entities (inspired from Anadiotis et al. (2021)).

of scientific publications, listed in a server such as PubMed, may report their relationships with organizations that fund their research, using the Conflict of Interest (CoI) field of papers they author, e.g., Dr. Alice consults for ABCPharma. Separately, within the paper itself, "Acknowledgments", "Funding statements" and a few other keywords indicate paragraphs where authors may prefix stated relationships between authors and organizations; in the figure, the article in PDF is converted into a JSON document where Dr. Alice thanks HealthStar. ConnectionLens also includes Information Extraction (IE) modules, which extract, from any leaf node in the data graph, Named Entities (People, Locations and Organizations) Anadiotis et al. (2022), as well as other types of entities that journalists find interesting: temporal moments (date, time), Website URIs, email addresses, and hashtags. We designate any of these pieces of information as *entities*, and we model them as extra nodes. For instance, in Fig. 1, organizations are shown as pink nodes, people as yellow nodes, and places as green nodes. Each entity is extracted from a leaf text node, to which it is connected by a dashed edge. When an entity is extracted from more than one text node, the edges connecting it to those nodes increase the graph connectivity, e.g., "Alice" is extracted from two nodes in the bibliographic notice, and once from the JSON resulting from the PDF. Knowing **all the relationships between medical experts, companies, and lobbies** acting for them interests a journalist that studies the authorization (or not) of new medical or chemical products by national or international authorities, since such decisions are often based on experts' opinions.

Goal: efficient, interactive exploration of entity connections Having in mind such an investigation, journalists are interested in *data paths ending in entity pairs of certain kinds* in the data graph, e.g., in Fig. 1, "how are people connected to organizations?". In Fig. 1, there are three such paths: between Alice and her

employer (connection highlighted in blue), Alice and ABCPharma (yellow highlight), and finally Alice and HealthStar (gray highlight). Each of these paths traverses internal nodes and edges with certain labels, and depicts a specific kind of relationship in the application domain. This small example features a single individual and a few organizations; actual investigative journalism datasets, in this study, or in the Paradise Papers project, are much larger and more complex. One way to explore them is via keyword search, looking for connections between specific company or person names, as in Anadiotis et al. (2021), Anadiotis et al. (2022). As an alternative, or as a first step towards discovering a dataset content, in this work we seek to **automatically enumerate and compute paths between pairs of entities**, and show them to the user, ranked according to their potential interestingness. When shown a set of paths, users may pick one to *further explore*: how many pairs of entities are connected by each path?, which entities are most frequent?, etc. Note that it is important to consider paths *irrespective of the edges directions in the data graph*. This is because, depending on how the data is modeled, we may encounter $x \xrightarrow{\text{boughtProperty}} y \xrightarrow{\text{locatedIn}} c$, or $x \xleftarrow{\text{hasOwner}} y \xrightarrow{\text{locatedIn}} c$; both paths could be interesting. Finally, we note that our approach is best suited to data graphs built for a specific application domain, e.g., "tax havens in Panama", or "subventions granted to projects around the city of Lyon", etc. In an universal-knowledge graph such as DBPedia, Wikidata, etc., the number of entity paths would be much higher, and other mechanisms (typically users specifying some terms of interest, etc.) would be needed to select a reasonable amount of paths to show them.

Challenges and contributions The analysis outlined above raises several challenges. First, while current IE tools achieve pretty good quality, experiments on several datasets recall that a few errors (false negatives, i.e., entities missed, and false positives, i.e., entities wrongly identified) subsist, leading to entities of a certain kind being (wrongly) considered as appearing in certain fields of the data. In turn, these entities may participate in many paths in the data; we say such paths are not *reliable*. Further, reliable paths may reflect more or less frequent (typical) relationships within a dataset. We call *strong* a path between named entities that has good support within the dataset. This can be seen as an indicator of a frequent relationship between the real-world entities which the dataset is about. Our main goal is to show users reliable and strong paths. Second, *materializing the entity pairs connected by such paths* may be very costly, if (*i*) the graph is large, and/or (*ii*) there are many paths, which happens if the data is complex/heterogeneous, and/or if we allow paths to traverse edges in both directions). Third, the large

number of paths may *overwhelm users*. Non-expert users, or users which are not familiar with the dataset structure, should not be required to state the paths that they would like to see; instead, the system should propose these paths.

Our method, named PathWays, addresses these challenges with the following contributions. (*i*) Towards identifying reliable paths, we added to ConnectionLens a *novel IE module*, based on OpenAI's ChatGPT API (Sec. 3). As we show, this significantly improves the extraction quality, avoiding numerous false positives that arise when using our previous IE module. (*ii*) We leverage this (state of the art) extraction quality, to assign to each path a *reliability* measure, which provides a first tool for ranking paths, when showing them to the user (Sec. 4). (*iii*) To break ties between paths whose reliabilities are very close, we introduce a novel structural metric called *path force*, based on graph node and edge cardinalities. This improves over our prior work Barret et al. (2023b) by eliminating user effort while keeping the quality of the proposed paths (Sec. 4). (*iv*) To speed up the entity paths materialization, we *recommend a set of views (subpaths) to materialize*, and *rewrite each path query using these materialized views* (Sec. 5). This allows to identify subpaths that appear in more than one path, and compute the corresponding data paths only once, greatly improving performance (Sec. 6). We then discuss related works (Sec. 7) and present some perspectives (Sec. 8).

2. From datasets to data graphs

In order to propose a general approach that works on any data format (e.g., RDF, JSON, XML, etc.), we leverage prior work Anadiotis et al. (2022), which builds a *data graph* out of each input dataset (Sec. 2.1). These graphs may be large, thus enumerating paths on them would be inefficient. Therefore, we rely on a more compact structure derived from this original graph as described in Barret et al. (2024), namely a *collection graph* (Sec. 2.2). Finally, we explain how we produce a single collection graph out of *several datasets* (Sec. 2.3), which journalists may need to exploit together in a particular investigation.

2.1. From a dataset to a data graph

We first show how ConnectionLens transforms datasets of various models into data graphs. Next, we show how Named Entities are identified in such data graphs; NEs are crucial to inter-connect data and better understand the dataset. Finally, we describe how we build a collection graph, a core structure for efficiently enumerating paths, and how we can compute it for several datasets at a time.

2.1.1. Dataset conversion in a graph

Given a set of datasets, ConnectionLens transforms them into a directed graph $G_0 = (N_0, E_0, \lambda_0)$ where N_0 is a set of nodes, E_0 is a set of edges connecting N_0 nodes and λ_0 is a function assigning a label l to each node and edge. We map each data model on G_0 as follows.

RDF graphs are naturally mapped on G_0: each subject, respectively object, is turned into a node and an edge labelled with the property is connecting them. An XML document corresponds to a tree, with element nodes having element and attribute children. XML elements may carry #ID attributes whose values uniquely identify them; other XML elements may carry #IDREF attributes, whose values act as "foreign keys" referring to other elements by their #ID value. ID-IDREF information can be supplied in an optional Document Type Description (DTD) or XML Schema (XSD); when these are not available, ID-IDREF links can be found by profiling Jiang and Naumann (2020); Abedjan et al. (2018) techniques, which we also implemented. In the graph representation of an XML document, ID-IDREF links lead to more edges between elements (thus, the graph is no longer a tree). HTML documents are similarly treated. JSON documents are also modeled as trees, where each map, array, and leaf value is a node and parent-child edges are connecting them. For relational datasets, a directed graph representation has been established in prior research focused on keyword search in relational databases Agrawal et al. (2002), and ConnectionLens also adopts it. Specifically, a node is created for each tuple in a relation; the node has an outgoing edge labeled with each attribute name from the relation, leading to a leaf node labeled with the respective attribute value. Attributes encapsulating foreign keys are treated differently: each foreign key relating a tuple $r \in R$ to a tuple $s \in S$, where R, S are two relations such that a foreign key goes from R to S, are modeled as arrows going from the node corresponding to a tuple $r \in R$, to the node corresponding to the respective tuple $s \in S$. A CSV dataset is ingested like a single relation. Property graphs are converted to ConnectionLens ones by creating one node for each PG node and edge, one node for each of their attributes, connecting together each node (edge) with its attributes via edges labeled after the attribute names, and representing each PG relation as a succession of two edges, from the PG relation source to the relation node, and from there to the PG relation destination. Document formats, e.g., PDF, Office formats, etc., are converted to JSON using existing libraries, then their content is ingested as JSON.

We stress that regardless of the original format, content is ingested in a *fine-granularity* graph, where each node has an (internal) ID given by ConnectionLens,

and a label that may also be empty; each edge also carries a label, possibly empty. We also store, for each node, the dataset from which it comes, and its type, e.g., whether it was an RDF literal, an XML element, etc.

2.1.2. Named Entity extraction

In Information Extraction (IE), a **Named Entity** (NE) is a real-world object, such as a Person, Place, Organization, Product, Event, etc., that can be denoted with a proper name. Named Entities can be extracted from text using Named Entity Recognition (NER) tools. A NER tool takes as input a string and yields a set of triples of the form $\langle NE, \tau, c \rangle$ where NE is a sequence of tokens from the input string, considered to be the label of an NE, τ is the type associated to that entity, and c is the confidence the extractor has in NE and τ.

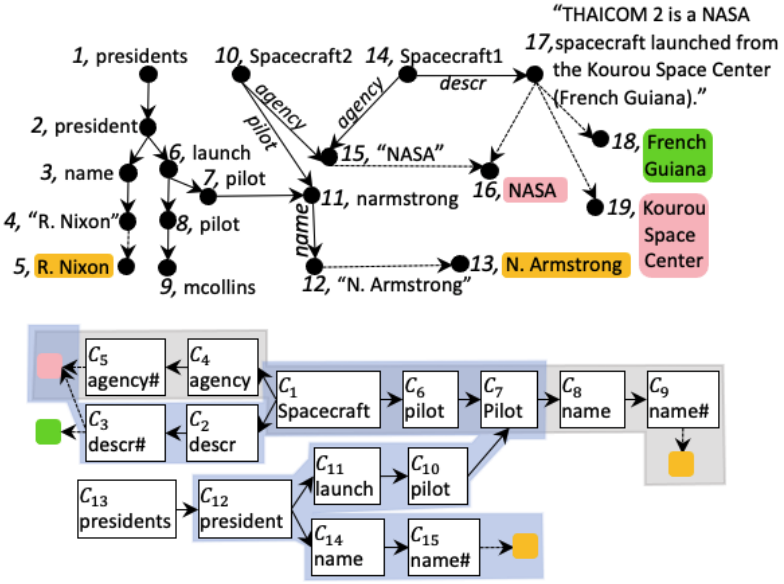

Figure 2: Sample data graph (top), and corresponding collection graph (bottom) on which paths linking entities are explored (highlighted areas).

Fig. 2 shows a running example on which we will rely below, which concerns NASA flights. The upper half of the figure shows the ConnectionLens data graph built out of two datasets: at left, an XML document describing presidents who attended spacecraft launches (tree with labeled nodes and unlabeled edges); at right, a (drastically simplified) sample of the NASA RDF dataset. We show next to each node its ID and label, and the label of each edge in *italic*, when not empty; for

readability, we replace each RDF URI with a short, readable label, e.g., Spacecraft1, *descr*, etc. Labels of text nodes (literals) are enclosed in double quotes. The nodes on colored background are extracted NEs: yellow for Person, green for Location, and pink for Organization entities. Each NE is connected to its parent(s) (string(s) from where it has been extracted) by a dashed line, e.g., the Organization node NASA (16) is a child of both strings "NASA" (15) and "THAICOM 2 is a NASA... (French Guiana)" (17).

Some entities can be detected with simple **pattern-based extractors**, when a regular expression describes the set of legal NE labels; this is the case for emails, hashtags, Twitter user mentions, URIs, and ConnectionLens recognizes them via pattern matching. Recognizing entities of other types, i.e., Person, Location, Organization entities, or dates, require a pre-trained language model. ConnectionLens Anadiotis et al. (2022) extracts such entities using two extractors, each of them is based on a **pre-trained language model**. The first one is based on the **Stanford NER** Manning et al. (2014), while the second one is based on the Deep Learning **Flair** NLP framework Akbik et al. (2019), pre-trained on the CoNLL-2003 news articles dataset.

While the Flair extractor is much more accurate than the Stanford NER one, it still leads to false positives (entities recognized where one does not really appear), as well as false negatives (actual entity occurrences missed by the extractor). *False positives negatively impact PathWays*, since they lead to erroneous extraction edges, which, in turn, lead to spurious paths between extracted entities. This increases the path evaluation effort, without bringing interesting results.

To solve this problem, one could retrain the Flair model for different corpora, but this is labor-intensive. Instead, to improve the path quality w.r.t. the paths found in Barret et al. (2023b), we have developed a new **NE extractor, leveraging OpenAI's ChatGPT 4**. Relying on this very large model has led to better-quality results, once properly prompted. We describe the new extractor in Section 3.

2.2. From a data graph to a collection graph

The collection graph is a compact representation of the data graph. It is based on an equivalence relation between the data graph nodes: the collection graph has exactly one node for each equivalence class of data graph nodes; further, whenever $n_1 \rightarrow n_2$ is an edge in the data graph, the collection graph comprises an edge $C_1 \rightarrow C_2$, where C_1, C_2 are the equivalence classes (also called collections) to which n_1, n_2 belong, respectively.

The most natural node equivalence relation differs across data models. Specifically, XML nodes we consider equivalent are elements with the same name; text

nodes that are children of equivalent elements; and values of same-name attributes of equivalent elements. For instance, in Fig. 2, the pilot nodes 7 and 8 are equivalent. In JSON, we consider equivalent nodes found on the same labeled path, from the JSON document root, to the node. A path is a concatenation of node and edge labels, separated by . (dots), where we assign special labels: μ to each map node, α to each array node, and ϵ to each empty node or edge label. For instance, in the JSON snippet [{"name": "Alice", "address": {"street": "Main", "city": "NY"}}] the path to "NY" is: $\alpha.\epsilon$.address.μ.city. In RDF, there are numerous ways to define node equivalence Cebiric et al. (2019). RDF collection graphs are built through the TypedStrong summarization method Goasdoué et al. (2020), working as follows. Whenever an RDF node has one or more types, all nodes with the same set of types are said equivalent (in RDF, a node can have several types, related or not, e.g., *Student* and *Employee*). For nodes without types, TypedStrong summarization relies on the properties (labels of incoming and outgoing edges) that the nodes have, by introducing a notion of *outgoing/incoming property cliques*: (*i*) two properties that a node has, are in the same outgoing clique, e.g., agency and pilot are in the same outgoing clique because they are both property of node 10 in Fig. 2, and also with descr because node 14 has agency and descr; incoming property cliques are symmetrically defined based on incoming properties; (*ii*) two nodes without types are equivalent if they have identical outgoing and incoming property cliques. For instance, nodes 10 and 14 (the two spacecrafts) are equivalent, since they have the same outgoing and incoming property cliques.

Fig. 2 (bottom) shows the collection graph corresponding to the data graph. We named the collections C_1, C_2, etc. Note that some data models have labeled edges, e.g., RDF, while others have (mostly) unlabeled edges, e.g., XML. For uniformity, we transform any labeled edge into a node and extend our summarization also to such nodes. In Fig. 2, collection C_6 contains the nodes introduced instead of the pilot edges in the RDF dataset. Collection names in our figure are only for ease of explanation (they are not required in our method). For collections such as C_1, with RDF nodes each with a different URI, we use an intuitive name, e.g., "Spacecraft".

Entity collection profiles Each leaf collection in the collection graph corresponds to a set of literals (strings), out of which various NEs may have been extracted. These collections' names end with a # to help distinguish them (e.g., C_5 agency#). For each such collection C, we compute an *entity profile* storing how many entities of each type were extracted from its string nodes. In Fig. 2, there are four such profiles, each shown as a box filled with the color of an entity type, e.g., the child of C_5 is pink reflecting the Organization entities extracted from agency values. In

practice, long text nodes often lead to multiple NEs extracted, of several types. Knowledge of which leaf collections contain which kind of entities will be crucial to help users explore the graph (Sec. 4).

2.3. From multiple datasets to a collection graph

Journalists often need to work with several datasets, e.g., a JSON collection of political tweets, the list of French mayors in XML, and an RDF graph of public investment in companies across France. When ingesting several datasets, we need to decide whether and how to interconnect them. (*i*) As per the RDF specification, when two or more RDF graphs feature a common URI (or literal), only one node should be created for that URI (or literal). In other words, RDF graphs are automatically merged. (*ii*) Because of the particular semantics of URIs, at any point, in any ConnectionLens graph, there is at most one labeled with any URI, even if the URI appears in a dataset of a model other than RDF. For instance, in Fig. 2, the node with ID 7, labeled pilot, has as child the URI narmstrong; this interconnects the XML and the RDF parts of the graph. All the other nodes are created independently from each other, each considered as part of exactly one dataset.

To obtain a single collection graph from a set of datasets, we proceed as follows. First, we build a separated collection graph from each dataset (as in Sec. 2.2). Then, whenever two collections C_1, C_2 from distinct datasets share at least a URI, we replace them by a new collection C_3, which contains the values of C_1 and C_2, and inherits all the incoming and outgoing edges of C_1 and C_2 in the collection graph they came from. Their original collection graphs are thus connected. This is how the collection C_7 in Fig. 2 has two incoming edges, one reflecting the XML hierarchy at the top left of the figure, and the other the RDF edges at the top right. Further, when datasets are ingested together in ConnectionLens for analysis, they typically feature *common NEs*. For instance, in Fig. 2, at the extreme left of the lower half, the pink-filled square denotes the collection comprising Organization entities extracted from the values of the descr and agency RDF properties; there is a single such collection, because they share the NASA Organization entity. Interesting data paths can thus be found *across* data sources.

3. ChatGPT-based Named Entity extractor

To improve the quality of NE extraction, we have extended ConnectionLens with a new extractor, leveraging OpenAI's ChatGPT v4. Its very large training corpus allowed us to expect good performance in detecting NEs in texts of various forms even without corpus-specific fine-tuning.

Please get each named entity you identify in the following string: 'XXX'. Return a table containing four columns, one for the named entity name, one for the named entity type you assigned to it, one for a category among (person, organization, location) that fits your type, and one with the confidence you have in the category assigned to the extracted entity where the confidence is a float value between 0 and 1. If no category fits your type or a sub-type of your type, set the category as OTHER. If no named entities of the expected types, answer NONE.

Figure 3: ChatGPT prompt for NE extraction (directive plus a string, denoted 'XXX').

We used ChatGPT in a question-answer mode. Each question we send, also called *prompt*, consists of a fixed directive, together with a string (text leaf in the dataset) in which we ask ChatGPT to identify named entities. For each entity found, ChatGPT also returns a type it considers the most suited for the entity in the context of the input string, as well as its confidence. Since ConnectionLens uses the NE types People, Organizations, or Locations (recall Sec. 2.1.2), our prompt instructs ChatGPT to identify entities of these types, or to an extra category OTHER. The best prompt we found appears in Fig. 3. It constrains the answer format to facilitate its integration in ConnectionLens. Just like the previous extractors (Sec. 2.1.2), this prompt is independent of the application domain. Our first prompts did not mention the OTHER category; we observed that this lead to some false negatives, i.e., some Person, Organization or Location entities (entity types we had explicitly asked for) were not returned. We believe introducing the OTHER category clarified the prompt semantics; it certainly lead to better-quality results.

For instance, Tab. 1 shows the NEs (with their attached information) found by ChatGPT in the string: *"Declaration of competing interest Y. Hu and A. Coates are the coinventors of the antibiotic resistance breaker technology, in particular the combination of the quinoline and tobramycin (patent granted). They were the first to test this combination against highly resistant Pseudomonas spp. They originated the concept and performed the background work upon which this work is based. A. Coates, Y. Hu and CP declare they have equity in Helperby Therapeutics who are developing HT61. CP is in receipt of a grant from Helperby Therapeutics to support Dr Richard Amison for the conduct of the in vivo aspect of this study. There are no other conflicts of interest to declare"*. This string appears in a Conflict of Interest statement of a PubMed article (recall our motivating scenario Anadiotis et al. (2021) illustrated in Fig. 1). As Tab. 1 shows, our prompt is effective in getting ChatGPT to perform high-quality extraction. We analyze its

Named Entity	Type	Category	Confidence
Y. Hu	Person	Person	0.95
A. Coates	Person	Person	0.90
antibiotic resistance breaker technology	Product	OTHER	0.70
quinoline and tobramycin	Product	OTHER	0.70
Pseudomonas spp.	Species	OTHER	0.8
Helperby Therapeutics	Company	Organization	0.85
HT61	Product	OTHER	0.60
Dr Richard Amison	Person	Person	0.80

Table 1: Sample ChatGPT NE extraction results.

performance in more depth in Sec. 6.2.

4. Paths between entities

In this section, we discuss categories of paths that users might be interested in, and how to ask for their input.

An important first observation is: by the way we built the collection graph, *to each dataset path corresponds a path in the collection graph*. For instance, consider the data path $13 \leftarrow 12 \xleftarrow{\text{name}} 11 \xleftarrow{\text{pilot}} 10 \xrightarrow{\text{agency}} 15 \rightarrow 16$ in Fig. 2. The collection graph features the corresponding path ■ $\leftarrow C_9 \leftarrow C_8 \leftarrow C_7 \leftarrow C_6 \leftarrow C_1 \rightarrow C_4 \rightarrow C_5 \rightarrow$ ■.

Further, *some paths in the collection graph correspond to no paths in the data graph*. For instance, the path $C_6 \leftarrow C_1 \rightarrow C_2$ does not correspond to any path in the data, because no spacecraft (part of the collection C_1) has *both* the agency and descr properties. Such paths are introduced by summarization, which compresses the graph structure with some information loss. In our example, the fact that a spacecraft has agency and pilot, another has pilot and descr is "simplified" into a collection C_1 that may have any combination of the three properties (represented by collections C_2, C_4, C_6). We accept this loss of information in exchange for the ability to work on the collection graph, much smaller than the data graph.

Based on the above, our approach is: (*i*) enumerate paths on the collection graph, (*ii*) identify the *interesting* paths as those that are both *reliable* and *strong* (see below), then (*iii*) turn each path into a query, and (*iv*) evaluate this query on the data graph, and show users the resulting actual data connections (if any).

4.1. Notations for entity paths in the collection graph

Each path between two entities is first, characterized by a **pair of entity types** of the form (τ_1, τ_2), where $\tau_1, \tau_2 \in \mathcal{E}$, with $\mathcal{E}$ being the set of supported entity types. $\mathcal{E}$ contains entity types such as Person, Location, Organization, Email, URI, Hashtag, Date, etc.

An entity path is also characterized by its **length**, i.e., the number of edges it contains. Depending on the application, interesting connections can be made by paths of different lengths; however, it appears likely that beyond a certain length, connections may become meaningless. Therefore, and also to control how many collection paths they want to inspect, users may specify a **maximum path length** L_{max}, whose default value we set to 10. Moreover, longer paths tend to be of a lower quality because it is very likely that they will contain non-reliable or weak collection edges (see Sec. 4.3 and 4.4), thus are often ranked low, thus not shown to users.

Each entity-to-entity collection path cp is of the form: $\square \leftarrow C_1 \leftrightsquigarrow C_2 \rightarrow \blacksquare$, where $\square, \blacksquare$ are two entity profiles, such that the first, respectively, the second, contains some entities of types τ_1, respectively, τ_2, and C_1, C_2 are value collections such as C_5 and C_9 in the example at the beginning of Sec. 4. The directions of the leftmost and rightmost edges are by convention always towards $\square, \blacksquare$, which represent entities. Let cp_0 denote the path $C_1 \leftrightsquigarrow C_2$.

4.2. Path analysis by directionality

We can classify entity paths according to the directions of the edges in $\leftrightsquigarrow$. Specifically, paths may be:

- *Unidirectional*, i.e., all cp_0 edges go from C_1 towards C_2, or the opposite;
- *Shared-sink*, i.e., cp_0 may contain a (collection) node C such that all edges between C_1 and C (if any) go from C_1 towards C, and all edges between C_2 and C (if any) go from C_2 towards C. A shared-sink path is $C_1 \rightarrow C_6 \rightarrow C_7 \leftarrow C_{10} \leftarrow C_{11}$.
- *Shared-root*, i.e., cp_0 may contain a (collection) node C such that all edges between C and C_1 (if any) go from C towards C_1, and all edges between C and C_2 (if any) go from C towards C_2. A shared-root path is $C_9 \leftarrow C_8 \leftarrow C_7 \leftarrow C_6 \leftarrow C_1 \rightarrow C_4 \rightarrow C_5$.
- *General*, i.e., the edges may be in any direction.

Unidirectional paths are quite rare. This is because entity-connecting paths must have at each end a node from which an entity is extracted. Most of the time, these are *two literal (string) nodes* (as opposed to internal nodes structuring the dataset). Literals have incoming edges, but not outgoing ones (other than those towards extracted entities); thus, *there is no unidirectional path from a literal to another*. However, in some RDF datasets, *NEs are extracted from URIs*, e.g., the triple https://dbpedia.org/Facebook $\xrightarrow{\text{locatedIn}}$ http://dbpedia.org/California is a unidirectional data path from an Organization to a Location. Similarly, shared-sink paths only occur when nodes in C_1 and C_2 have outgoing edges, and NEs appear in their labels; this only happens in RDF URIs.

4.3. Path reliability

As previously stated, each entity path in the collection graph reflects zero or more paths in the data graph. Data graph edges can be divided into: (*i*) structural, which originate in the way the data is organized in the input graph; in our approach, we consider such edges as certain, or fully reliable; and (*ii*) extraction edges, which connect a NE to the string from which it had been extracted. In turn, an extraction edge can reflect a true positive (the entity is correctly extracted, i.e., most human users would agree that the entity of the respective type is present in the string), or a false positive (a human user would not consider the entity is present there). In the collection graph, we call *extraction edge* an edge corresponding to one or more data graph extraction edges.

Each entity path has at least two extraction edges, □ $\leftarrow C_1$ and $C_2 \rightarrow$ ■; if the path is either shared-sink or general, it may contain other extraction edges. In Fig. 2, the path ■ $\leftarrow C_3 \rightarrow$ ■ $\leftarrow C_5 \leftarrow C_4 \leftarrow C_1 \rightarrow C_6 \rightarrow C_7 \rightarrow C_8 \rightarrow C_9 \rightarrow$ ■ in the collection graph exhibits four extraction edges, including the two adjacent to the entity collection ■.

Examining several datasets, we noted situations when *all* the data extraction edges behind a given collection graph extraction edge e correspond to false positives, due to IE errors. For instance, in PubMed bibliographic data, chemical acronyms in article titles were mistakenly extracted as being Organizations. In more subtle cases, people names were extracted from (*i*) titles; two among a few thousands articles were scientists' obituaries thus had their name in the title; (*ii*) affiliations, when a research lab, institution, or a street is named after a person. In such cases, a person name is technically present, but since most titles, and most affiliations, do not feature people, we consider that the collection-level extraction edge is not *reliable*.

Formally, for each collection graph extraction edge e of the form $C \rightarrow \blacksquare$, where $\blacksquare$ corresponds to NEs of a specific entity type τ, we compute the **reliability of** e, denoted e_{rel}, as: the fraction of data graph nodes in C having at least one extracted entity child of type τ. Further, we compute the **reliability of an entity path** as the minimum value of e_{rel} over all the extraction edges e present in the path. Using the minimum to aggregate extraction edge reliabilities is a conservative choice, which penalizes a path according to its least reliable edge. In our experiments, this choice gave good results; indeed, even a single unreliable edge in a collection graph path may make it meaningless.

4.4. Path force

Beyond directionality and reliability, entity paths can also be analyzed based on the numbers of edges adjacent to graph nodes. ConnectionLens Anadiotis et al. (2022) attaches to each edge e of the original data graph, having a non-empty label a, a measure called *specificity*. Let e be the edge $n_1 \xrightarrow{a} n_2$ for some nodes n_1, a, n_2. The *specificity* of e, denoted e_s, is computed as $2/(N_{1,a} + N_{2,a})$, where $N_{1,a}$, $N_{2,a}$ are the numbers of edges labeled a outgoing, resp. incoming, n_1, resp. n_2. The highest $N_{1,a}$ and/or $N_{2,a}$, the lowest e_s. For instance, the specificity of each agency edge in Fig. 2 is $2/(1 + 2) = 2/3$. For our purposes, we extend specificity to unlabeled edges as follows: the specificity of an edge $n_1 \xrightarrow{\epsilon} n_2$ is $2/(1 + n_{1,2})$ where $n_{1,2}$ is the number of ϵ (empty-label) edges outgoing n_1, towards nodes having the same label as n_2. For instance, the specificity of the edge between nodes 6 and 7 is also $2/3$. In Anadiotis et al. (2022), edge specificity has been used as an ingredient for scoring connections (paths or trees) in the original graph, returned when users search the graph using keywords. Indeed, low-specificity edges can be seen as "weakening" connections, e.g., when a person has 1 spouse and 200 friends, intuitively, an edge (thus, path) going from the person to a friend is weaker than one going from the person to the spouse.

In Barret et al. (2023b), we leveraged data edge specificity as follows. (*i*) In the collection graph, the edges with a non-empty label, connecting nodes from two equivalence classes lead to a collection, e.g., agency triples lead to C_4. To this *collection* is attached *the average specificity of all the data edges it comes from*, e.g., to C_4 corresponds $2/3$. Empty-label edges connecting graph nodes from two equivalence classes lead to an edge in the collection graph, e.g., $C_{11} \rightarrow C_{10}$. To an edge between collections is attached the average specificity of the original edges. (*ii*) *Paths with low-specificity collections or edges were pruned.* Specifically, users were first asked to state how many low-specificity collections and collection edges they are willing to review, then shown the lowest-specificity ones,

each of which they can validate or invalidate. Then, we only enumerated paths that did not traverse invalidated collections. This approach allowed to control the effort required from users; it is also more accurate than just invalidating collections whose specificity is below a given threshold, which could lead to decisions suitable for some collections but unsuitable for others. However, this approach of Barret et al. (2023b) has its drawbacks. On one hand, it requires user effort; on the other hand, low-specificity collections and collection edges, that do not make it to the users' inspection, may be preserved, leading to weak paths.

Towards avoiding these limitations, we change our approach, as follows. First, we *no longer require users to inspect structural metrics*. Second, we *use structural metrics to rank paths, instead of pruning them*. Third, we *compute different structural metrics*, as discussed below.

Path force Let n_i, n_j be nodes in the normalized data graph, belonging to the collections C_i, C_j, respectively. We define the **(data) edge cardinality** e_{card} of a data edge $e = n_i \rightarrow n_j$ such that $n_i \in C_i, n_j \in C_j$ as the number of data edges outgoing n_i and ending up in a node n_z such that $n_z \in C_j$. This differs from specificity in several respects. (*i*) Cardinality is asymmetric, i.e., it only considers how many edges exit a node in C_i, not how many enter a particular node in C_j. This seems simpler and more intuitive. (*ii*) Cardinality interprets the neighborhood of n_i through the prism of the collections C_i, C_j to which n_i, n_j belong. Specificity had no awareness of collections, and only focused on edges exiting n_i and entering n_j. Next, we define the **collection edge force** $f(C_i \rightarrow C_j)$ of a collection edge as the inverse of the maximum cardinality among all data edges represented by the collection edge; this number is in $(0, 1]$. Taking the inverse of the maximum cardinality penalizes the existence of even one node $n_i \in C_i$ having a large number of edges to nodes from C_j. Averaging specificities as in Barret et al. (2023b) allowed to smooth the impact of such nodes. We prefer the inverse of the maximum cardinality since we consider a C_i node with many edges to C_j signals that connections $C_i \rightarrow C_j$ may be not too selective for a C_i node, i.e., one *could* have hundreds of friends, or written hundreds of papers, even if most people do not. In contrast, one always has a single birth country, at most one or a few spouses over a lifetime, etc. Finally, we define the **path force** $F(p)$ of a path p as $F(p) = \prod_{C_i \rightarrow C_j \in p} f(C_i \rightarrow C_j)$. This combines all forces along the path, penalizing multiple and/or low values. It also penalizes paths containing many edges whose force is below 1.

4.5. Putting it all together: collection path enumeration

Our approach for collection path enumeration is as follows. First, we ask users which two entity types they want to connect, i.e., select τ_1, τ_2. Next, users can set

the maximum path length L_{max} they are interested in. Next, based on the collection graph, we *enumerate all the paths connecting entities* of types τ_1 and τ_2. We use a simple in-memory dynamic programming algorithm, as the collection graph is much smaller than the data. Then, we compute the reliability (Sec. 4.3) and force (Sec. 4.4) of each path thus enumerated: we first compute the reliability of each extraction edge and the force of each non-extraction one, then combine them into reliability and force values for the paths. We then inform the user: "There are N_{uni} unidirectional paths, N_{sink} shared-sink paths, N_{root} shared-root paths, and N_{gen} general paths between entities of type τ_1 and τ_2." We show the paths sorted, first, by reliability (truncated to just two decimals), then by force. The user can then chose a set of collection paths to materialize, e.g., "only the shared-root ones", or "the ones ranked in the top-20", or "just those involving specific internal node labels", etc. How paths are materialized will be discussed in Sec 5.

5. Materializing data paths

At this point, we have a set of *collection paths*, which must be transformed into queries and evaluated *on the data graph*. The results of each such query are shown to users as a table: the first and last attribute of such a table comprise entities of type τ_1, τ_2, while the intermediary attributes are the nodes and edges connecting these entities in the data graph. For instance, let τ_1 be Person, τ_2 be Organization: the light-blue, respectively, light-gray background shapes in Fig. 2 materialize the two paths which, in this graph, connect the pink child of C_5 (■) with the yellow children (■) of C_9, respectively, C_{15}.

5.1. From a collection path to a query over the data graph

Each collection path translates into a chain-shaped conjunctive query. For instance, the path on gray background in Fig. 2, going through C_5 and C_9, becomes:

$$q_1(\bar{x}) :- \; n(x_1, \tau_{\text{Org}}, \blacksquare), e(x_2, x_1, \_), n(x_2, \_, C_5), e(x_3, x_2, \mathsf{agency}), n(x_3, \_, C_1), e(x_3, x_4, \mathsf{pilot}), n(x_4, \_, C_7), e(x_4, x_5, \mathsf{name}), n(x_5, \_, C_9), e(x_5, x_6, \_), n(x_6, \tau_{\text{Person}}, \blacksquare)$$

This query refers to two relations: $n(ID, type, equiv)$, describing nodes, with the last attribute denoting their equivalence class, and $e(s, d, label)$, describing edges between nodes s and d and carrying a certain label. Each x_i is a variable; $\bar{x}$ in the query head denotes all the x_i variables, $1 \leq i \leq 6$. We use _ to denote a variable which only appears once, in a single query body atom. Finally, τ_{Org} and τ_{Person} denote the node types of extracted Organization, respectively, extracted Person entity. Similarly, the blue-background collection path translates into:

$$q_2(\bar{x}) \text{ :- } n(x_1, \tau_{\text{Org}}, \blacksquare), e(x_2, x_1, \_), n(x_2, \_, C_3), e(x_3, x_2, \mathsf{descr}), n(x_3, \_, C_1), e(x_3, x_4, \mathsf{pilot}), n(x_4, \_, C_7), e(x_5, x_4, \_), n(x_5, \_, C_{11}), e(x_6, x_5, \_), n(x_6, \_, C_{12}), e(x_6, x_7, \_), n(x_7, \_, C_{14}), e(x_7, x_8, \_), n(x_8, \_, C_{15}), e(x_8, x_9, \_), n(x_9, \tau_{\text{Person}}, \blacksquare)$$

Each of these queries can be evaluated through any standard graph database. However, evaluating dozens or hundreds of path queries on large graphs can get very costly. Further, since we do not know which paths may result from the user choices, we cannot establish path indexes beforehand.

View-based optimization To address this problem, we propose an optimization, based on the observation that *queries resulting from collection paths may share some subpaths*. For instance, the subquery $s(x_3, x_4)$:- $n(x_3, \_, C_1), e(x_3, x_4, \mathsf{pilot}), n(x_4, \_, C_7)$ is shared by $q_1(\bar{x})$ and $q_2(\bar{x})$. Therefore, we decide to (*i*) evaluate s and store its results; (*ii*) rewrite $q_1(\bar{x})$ and $q_2(\bar{x})$ by replacing these atoms in each query, by a single occurrence of the atom $s(x_3, x_4)$. The next sections formalize this for larger query sets, also showing how to handle different *alternatives* that may arise as to which shared subpaths to materialize.

5.2. Enumerating candidate views

A first question we need to solve is enumerating, based on a set Q of path queries, the possible subqueries that we could materialize, and based on which we could rewrite some workload queries.

Let $q \in Q$ be a path query: it is an alternating sequence of node (n) and edge (e) atoms. We denote by n_q the number of edge atoms, then the number of node atoms is $n_q + 1$. We denote by n_Q the highest n_q over all $q \in Q$.

Without loss of generality, our first heuristic **(H1)** is: we only consider **connected subpaths** of q as candidate subqueries. If q is of the form $q(\bar{x})$:- $n_1, e_1, \ldots, e_{n_q}, n_{n_q+1}$, each connected subpath of q, denoted sq, is determined by two integers $1 \leq i \leq n_q$, $i < j \leq n_q + 1$, such that $sq(x_i, x_j)$:- $n_i, e_i, \ldots, n_j, e_j$, and x_i, x_j are the IDs of the nodes in the atoms n_i, n_j, respectively. We denote by $q|^{i,j}$ the subquery of q determined by the positions i, j. For instance, when q_1 is the sample query in Sec. 5.1, $q_1|^{3,4}$ is the subquery $s(x_3, x_4)$ introduced there. Considering connected (cartesian-product free) candidate views is common in the literature, too (see Sec. 7).

Each query $q \in Q$ has $O(n_q^2)$ connected subpaths, that can be easily enumerated from q's syntax. A second heuristic **(H2)** we adopt is: we only consider **shared subpaths**, that is, those subpaths s for which there exist $q', q'' \in Q$, $q' \neq q''$, and integers i', j', i'', j'' such that $s = q'|^{i',j'} = q''|^{i'',j''}$, possibly after some variable renaming. For the queries q_1, q_2 in Sec. 5.1, the subquery $s_{3,4}$ is $q_1|^{3,4}$ and also

$q_2|^{3,4}$. (H2) restricts the number of candidate views from $|\mathcal{Q}| \times n_Q^2$ to a number that depends on the actual workload $\mathcal{Q}$, and which decreases when $\mathcal{Q}$ paths look more like each other. Another interest of (H2) is: the benefit of using a view v to rewrite one query q is likely offset by the cost of materializing v; actual performance improvements start when v is *used twice (or more)*, which is exactly the case for subqueries shared by several $\mathcal{Q}$ queries.

Our third heuristic **(H3)** is: among the possible subqueries shared by two queries q', q'', **consider only the longest ones**. That is, if s_1, s_2 are two shared subqueries of q' and q'' such that $n_{s_1} > n_{s_2}$, do not consider the subquery s_2.

Our heuristics (H1), (H2), (H3) lead to **building the candidate view set $\mathcal{V}$** as follows. For each pair of distinct queries (q', q'') where $q', q'' \in \mathcal{Q}$, add to $\mathcal{V}$ the longest, shared, connected subqueries of q' and q''. The complexity of this algorithm is $O(|\mathcal{Q}|^2 \times n_Q^2)$, while $|\mathcal{V}|$ is in $O(|\mathcal{Q}|^2)$.

5.3. Selecting materialized views and rewriting path queries

Knowing the path queries $\mathcal{Q}$ and the candidate view set $\mathcal{V}$, we need to determine: a set $\mathcal{M} \subseteq \mathcal{V}$ of views which we actually materialize, in order to rewrite some $\mathcal{Q}$ queries. We collect the rewriting of each such queries in $\mathcal{R}$. The decision to materialize a view incurs a *cost*, since the view data must be computed and stored. We denote $cost(\cdot)$ the cost of evaluating a view (or query), and assume it can be estimated without actually evaluating the view (query). Materializing a view is more attractive if (*i*) rewritings using it reduce significantly query evaluation costs, and (*ii*) its own materialization cost is small.

In the most general case, a query could be rewritten based on any number of views, and also involving the base graph. For instance, query q_1 from Sec. 5.1 could be rewritten as: $q_1|^{1,3} \bowtie q_1|^{3,4} \bowtie q_1|^{4,6}$, where each $\bowtie$ denotes a natural join, on the variables x_3, respectively, x_4. However, enumerating all such alternatives makes the query rewriting problem NP-hard Halevy (2001). Instead, we adopt another pragmatic heuristic **(H4)**: **rewrite each query using not more than one view**. This simple choice keeps the view selection complexity under control, all the while providing good performance.

Algorithm 1 depicts our **greedy** method for finding $\mathcal{M}$ and $\mathcal{V}$. It computes the *benefit* of each view v for each query that may be rewritten using v, as well as the cost of v. In a greedy fashion, it decides to materialize the view v_{max} maximizing the *overall* benefit (for all $\mathcal{Q}$ queries), and uses it to rewrite all queries whose evaluation cost can be reduced thanks to v_{max}, via the rewriting $r_{q,v_{max}}$. These queries are then removed from $\mathcal{Q}$, the benefits of the remaining views are recomputed over

Algorithm 1: Selecting views to materialize and the respective view-based rewritings

Input : Queries Q, candidate materialized views $\mathcal{V}$, cost estimation $cost(\cdot)$, per-tuple CPU cost
Output: Materialized views $\mathcal{M}$ and rewritings $\mathcal{R}$ for some Q queries

```
1  M ← ∅; R ← ∅
2  while V ≠ ∅ do
3      for v ∈ V do
4          ben(v) ← 0; cost(v) ← cost to compute and store the view v
5          for q ∈ Q, q can be rewritten using v via the rewriting r_{q,v} do
6              ben(v, q) ← the cost of evaluating q directly on the graph,
                 minus the cost of evaluating q based on v, through the
                 rewriting r_{q,v}
7              ben(v) ← ben(v) + ben(v, q)
8      (v_max, b_max) ← a view v_max maximizing ben(v) − cost(v), and its benefit
9      if b_max − cost(v_max) < 0 then
10         exit
11     Add v_max to M
12     for q ∈ Q, q can be rewritten using v_max do
13         if ben(v_max, q) > 0 then
14             Add r_{q,v_max} to R
15             Remove q from Q
16     Remove v_max from V
```

the diminished Q, and the process repeats until no profitable view to materialize can be found.

Estimating costs Algorithm 1 needs to compute: (*i*) $cost(\cdot)$, the cost to evaluate a query q or materialize a view v; (*ii*) $r_{q,v}$, the rewriting of q using a view v; and (*iii*) $cost(r_{q,v})$, the cost of such a rewriting. All these costs must be estimated *before* any query or view results are computed. We do this as follows. For (*i*), we *use the cost estimation of the graph data management system* (GDBM, in short) *storing the graph*. Our implementation relies on PostgreSQL, whose `explain` command returns both the estimated number of results of certain query (or view), denoted $size(q)$, and the cost of computing those results. For (*ii*), recall (Sec. 5.2)

that when v is used to rewrite q, v is a subpath of q, thus there exist i, j such that $v = q|^{i,j}$. The rewriting $r_{q,v}$ is easily obtained by replacing, in the body of q, the atoms from the ith to the jth, with the head of v. Handling (*iii*) is more complex than (*i*). This is because the cost of a query (or view) is estimated based on *statistics* the GDBM has about the stored graph. In contrast, the GDBM cannot estimate cost of $r_{q,v}$, because *v has not been materialized* yet, thus the GDBM cannot reason about v like it does about the graph. To compensate, we proceed as follows: we compute the cost of reading the hypothetical view v_{max} from the database, by multiplying $size(v_{max})$, the estimation of the view size, with the per-tuple CPU cost (PostgreSQL's own CPU_TUPLE_COST); then, we estimate the cost of $r_{q,v_{max}}$ as this reading cost plus the cost of estimating the parts of q not in v_{max} plus the cost of joining v_{max} with these (one or two) remaining query parts. We estimate the cost of each such join by adding their input sizes. This reflects the fact that modern databases feature efficient join algorithms, such as memory-based hash joins, whose complexity is linear in the size of their inputs.

Complexity Algorithm 1 makes at most $O(|\mathcal{V}| \times |Q|)$ iterations, which can be simplified into $O(|Q|^3)$. Forming a rewriting takes $O(n_Q)$, bringing the total to $O(|Q|^3 \times n_Q)$.

Impact of heuristics As previously discussed, (H1) is universally adopted in the literature: no candidate view features cartesian products. (H2), imposing that views benefit at least two queries, preserves result quality, i.e., cost savings, under every *monotone cost model*, ensuring that the cost of evaluating a query q is at least that of evaluating s, when s is a subquery of q. In contrast, (H3) and (H4) may each divert from the globally optimal solution. However, as our experiments show, our chosen rewritings perform well in practice, and the algorithm itself is very efficient.

6. Experimental evaluation

Our approach is fully implemented in Java 11, on top of ConnectionLens Anadiotis et al. (2021, 2022) which builds the data graph (Sec. 2.1) and Abstra Barret et al. (2022, 2024) which builds the collection graph (Sec. 2.2); these are stored in PostgreSQL. We experimented on a Linux server with an Intel Xeon Gold 5218 CPU @ 2.30 GHz and 196GB of RAM. We used PostgreSQL v9.6. Our system is available at: `https://team.inria.fr/cedar/projects/abstra/pathways/`. Our evaluation seeks to answer the two following questions: (*i*) *how does the ChatGPT-based entity extractor compare with the previous best one available in ConnectionLens?* (Sec. 6.2); (*ii*) *how are NEs connected in each*

dataset? (Sec. 6.3); (*iii*) *how efficient is our multi-query optimization algorithm in reducing the time to evaluate paths queries over the data graph?* (Sec. 6.4); (*iv*) *how do reliability and force vary in our datasets?* (Sec. 6.5); and (*v*) *how efficient is our path evaluation on top-ranked paths*? (Sec. 6.6).

6.1. Datasets and settings

We present experiments on an XML, an RDF, two JSON, and a relational dataset. They all come from real-life applications (as opposed to synthetic) to stay close to application needs, and to ensure realistic Named Entities (NEs). Indeed, synthetic datasets are often generated with an interest on structure, while the leaf (text) values lack interesting information.

We used an **XML PubMed** subset describing scientific articles from PubMed, a database of biomedical publications. Each article is described by its title, journal, link, year, DOI, keywords list and authors list. Authors are identified by their name and their affiliation; they may declare their conflicts of interest in the `<COIStatement>` tag. We used the **RDF Nasa** dataset, describing NASA flights, spacecrafts involved in launches, related space missions and the participating agencies. Next, we used the **JSON YelpBusiness** dataset where each business has an id, a name, an address, a city, a state, a postal code and coordinates (latitude and longitude). It also has a set of categories, e.g., bakery, shoe store, etc., and a set of attributes, e.g., `acceptCreditCards` (the latter may be deeply nested). They also received reviews from customers modeled as a number of stars (from 0 to 5) and the number of reviews. **YelpBusiness4**, 4 times larger than YelpBusiness, allows studying the scalability of our algorithm. Finally, we used a subset of the **relational IMDB** dataset describing movies, actors, and their roles in the cast. This dataset is part of a benchmark Coffman and Weaver (2014) for keyword search on relational databases, a context where the database is seen as a graph, just like we do in this work. Tab. 2 shows for each dataset: the number of nodes $|N|$ and edges $|E|$ in the graph resulting from ingesting the data; the numbers of extracted NEs $|\tau_P|$, $|\tau_L|$, $|\tau_O|$ with the Flair extractor and the minimum edge specificity $min(e_s)$. Without loss of generality, we experiment with the NE types Person, Location, Organization, whose types are denoted τ_P, τ_L, τ_O, respectively. Note that the minimum edge specificity is attained by the IMDB dataset, on a Primary Key-Foreign Key data edge $\texttt{cast\_info} \xrightarrow{\texttt{role\_id}} \texttt{role\_type}$ (recall Sec. 2). Table `cast_info` contains the people involved in a movie, each under one among 11 role types, including: actor, actress, cinematographer, composer, costumer designer, etc. Thus, `cast_info`, has a foreign key toward `role_type` that describes the roles. The role "actor" is used by 386,123 tuples in the `cast_info` table, thus

Dataset name	$\|N\|$	$\|E\|$	$\|\tau_P\|$	$\|\tau_L\|$	$\|\tau_O\|$	$min(e_s)$
PubMed	63,052	89,710	5,993	2,151	5,096	0.001
Nasa	59,408	128,068	634	690	4,530	0.0002
YelpBusiness	57,963	61,627	322	427	1,437	0.001
YelpBusiness4	229,949	247,074	1,099	1,230	4,199	0.0002
IMDB	8,858,267	11,288,336	300,981	25,080	66,389	0.000005

Table 2: Dataset overview.

the node corresponding to the actor tuple in `role_type` has as many incoming edges, leading to a very low specificity for each of them.

6.2. Performance of ChatGPT entity extraction

We have analyzed the novel ChatGPT-based NE extractor introduced in this work (Sec. 3), from the angle of: speed (it is a remotely provided service, whose invocation requires remote calls), financial costs incurred, and results quality.

Tab. 3 shows, for 6 strings taken from the experimental datasets (Sec. 6.1): $|s|$, the number of characters in the string; T_{extr}^{Flair}, resp. T_{extr}^{GPT}, the Flair, resp. ChatGPT, time (in seconds) to send the extraction query, wait for the service answer, and retrieve it through a `java.net.HttpURLConnection`; $|Flair|$, resp. $|GPT|$, the number of NEs found by each extractor in the string; $|t_{cont}|$, the number of context tokens in the ChatGPT prompt; and $|t_{gen}|$, the number of tokens in the ChatGPT output.

With respect to **speed**, we note that Flair extraction time increases with regards to the input string size. On the other side, ChatGPT brings a comparatively huge overhead *per connection* (or per string sent to the extractor). Thus, T_{extr}^{GPT} times are much higher than T_{extr}^{Flair}, even if extraction itself is not longer than with Flair (which can be tested by sending the same query to the ChatGPT free online assistant). While T_{extr}^{GPT} will vary depending on the caller's internet connection and many other factors hard or impossible to control (where is the actual ChatGPT server located, its load, etc.), the slowdown incurred by the remote connection is likely to occur in all settings.

For what concerns **financial costs** (non-existent for our Flair extractor), GPT-4 use incurs around $0.03/1K context (or prompt) tokens and $0.06/1K generated tokens according to OpenAI prices. In total, extractions in Tab. 3 used 1,618 context tokens and 680 generated tokens, leading to a cost of $0.08.

To compare **extraction quality**, we first present an analysis *per value collections*, on the PubMed and Yelp datasets; in each such collection, as humans, we

String	|s|	T_{extr}^{Flair}	$\|Flair\|$	T_{extr}^{GPT}	$\|GPT\|$	$\|t_{cont}\|$	$\|t_{gen}\|$
s_1	13	0.022	1	2.465	1	126	37
s_2	16	0.020	1	2.208	1	128	40
s_3	80	0.067	3	5.171	5	140	119
s_4	125	0.106	5	6.503	6	148	114
s_5	673	0.371	8	10.741	9	267	194
s_6	3,191	1.931	10	6.789	8	809	176

Table 3: Flair and ChatGPT-based extractors time (in seconds) and cost analysis on sample strings.

have a very solid intuition of *what kind of entities to expect*, against which we can measure the extractors (Sec. 6.2.1). Next, we compute their agreement and disagreement globally, using the PubMed dataset, since it is rich in entities and has also quite a complex structure (Sec. 6.2.2). Finally, we inspect a set of concrete examples, to better understand the reasons for disagreement (Sec. 6.2.3).

6.2.1. Entities extracted from various collections

In Tab. 4, we compute for each dataset the number of Person, Location and Organization NEs found in leaf collections by each extractor. In the YelpBusiness dataset, in `address`, `city` and `state` values, ChatGPT finds only Locations, and finds a good number of them. In contrast, Flair (*i*) finds much fewer locations, e.g., 65 instead of 3,999 in `state` values, and (*ii*) finds up to hundreds of wrong Person and Organization in these attributes. In the PubMed dataset, in journal titles, ChatGPT (*i*) finds more Organization NEs than Flair and (*ii*) does not confuse text mentions of people, e.g., "healthcare professionals" and "family doctors", with Person NEs. In author names (`name`), ChatGPT recognizes 40 more authors than Flair. Finally, in article titles, ChatGPT finds much less NEs than Flair, because it is not misled by mentions of Locations, or substances that Flair might consider Organizations.

6.2.2. Extractor agreement and disagreement

Tab. 5 quantifies agreement and disagreement between the Flair and ChatGPT extractors, for the PubMed dataset.

Let us call "extraction outcome" the three entity types Person, Location, Organization, together with "no entity", denoting that no NE has been found in the string. The table has a line for each extraction outcome using Flair, and a column for each extraction outcome using GPT. Thus, each cell counts the number of strings on which a certain combination of extraction outcomes occurred. For

	Flair			ChatGPT		
Collection name	τ_P	τ_L	τ_O	τ_P	τ_L	τ_O
YelpBusiness						
name#	231	96	195	-	1	3,998
address#	426	101	21	-	3,850	-
city#	237	662	14	-	3,999	-
state#	-	65	-	-	3,999	-
PubMed						
Name#	5,963	-	-	6,004	-	-
Affiliation#	193	17,715	14,665	450	14,074	15,385
ArticleTitle#	15	123	339	58	-	60
JournalTitle#	9	88	300	-	91	463
CoiStatement#	152	16	330	143	5	192

Table 4: NE counts per type for Flair and ChatGPT extractors on YelpBusiness and PubMed datasets.

	GPT Person	GPT Location	GPT Organization	GPT no entity
Flair Person	**5,913**	6	11	98
Flair Location	25	**1,088**	507	905
Flair Organization	36	141	**2,988**	1,797
Flair no entity	101	1,335	1,233	–

Table 5: Comparison of Flair and ChatGPT sets of extracted entities.

instance, in 5,913 strings, Flair and GPT found the exact same Person entity; however, in 6 cases where Flair found a Person, GPT found a location instead; in 11 cases, GPT found an Organization; and in 98 cases, GPT did not find any entity.

Tab. 5 shows that **agreement is significant** (entity numbers in bold on the diagonal), and **very frequent for Person** entities. Also, entities recognized as Person by one extractor and of another type by the other are very rare. **ChatGPT disagrees more strongly** with Flair over **Flair's Organizations**, frequently considering that no entity at all exists with the same label. The same holds about **ChatGPT's Locations**: Flair finds no entity with the same label, almost as often as it agrees with ChatGPT. Such cases, when Flair finds an entity and ChatGPT does not, reflect: on one hand, exactly Flair's false positives that we seek to avoid, in order to increase edge and path reliability; on the other hand, extractor disagreements on the tokens to include in an entity label, as we exemplify below.

6.2.3. Fine-grain analysis of some examples

Flair Person entities not found by ChatGPT include "Claudin-7b", "Cytochrome" and "Claudin-h", which are all proteins. Flair was likely confused by the capitalization, and wrongly extracts them as Person from paper titles and abstracts. Other

mistakenly extracted Person entities include people names in street names such as Peter Henry Rolfs (PERS) in *"Av. Peter Henry Rolfs, 36570-900 Viçosa"*. These mistaken Flair extractions are made on paper titles and author affiliations, leading to unreliable extraction edges in the collection graph, and thus, to unreliable paths. ChatGPT's better training gives it an advantage here.

Flair Location entities not found by ChatGPT are mostly due to different allocations of tokens in entity labels. For instance, in the string "*Institute for Cancer Outcomes and Survivorship, University of Alabama at Birmingham, Birmingham, Alabama, USA*", Flair identifies: Institute for Cancer Outcomes and Survivorship (ORG), University of Alabama (ORG), Birmingham (LOC), Alabama (LOC) and USA (LOC) (five entities), while ChatGPT extracts four: Institute for Cancer Outcomes and Survivorship (ORG), University of Alabama at Birmingham (ORG), Birmingham, Alabama (LOC), and USA (LOC). In our table, this leads to "Alabama" and "Birmingham" counting as two Flair Locations not found by ChatGPT (and symmetrically, "Alabama, Birmingham" counts as a ChatGPT Location not found by Flair). Due to the presence of many affiliation strings in our dataset, such disagreements are frequent in the table cells involving Locations and/or Organizations. In these cases, we found that ChatGPT's choice of entity labels is better. For instance, "Birmingham, Alabama" is more specific than "Birmingham" (the latter exists in many places, among them also UK, etc.). Other Flair-extracted Locations are clearly incorrect, e.g., from "*Av. Professor Egaz Moniz, Lisboa 1649-028, Portugal"*, it extracts Av. (LOC).

Flair Organization entities not found by ChatGPT are mostly due to similar errors (Locations and Organizations competing for tokens). They also include other obvious errors, e.g., Ag (ORG) (silver), Critique of the Literature (ORG), Lolium perenne L. (ORG) (a plant), Drs. (ORG) (doctors), etc.

ChatGPT Person entities not found by Flair include, e.g., *"Antonio González"*, (Spanish name, probably under-represented in Flair's training set), *"John A. Reif, Jr"* (full name plus the "Jr" suffix, probably also rare in the training set), etc.

ChatGPT Location entities not found by Flair include: "Varese, Italy" (Flair found "Varese" and separately "Italy", see discussion of "Birmingham, Alabama"

above), "3-5-7 Tarumi" (address without the city, Japanese format), and numerous addresses including zipcodes. ChatGPT has better knowledge of international addresses, e.g., correctly identifying a Location in: the Korean address *"San 65, Bokjeong-Dong, Sujeong-Gu, Seongnam City, Gyeonggi-do, 461-701, South Korea"*; the Japanese address *"Yoshida, Sakyo-ku, Kyoto 606-8501, Japan"*; and the Polish address "*ul. Niezapominajek 8, 30-239 Krakow, Poland*", where Flair only accepted the city and/or the country.

ChatGPT Organization entities not found by Flair are again mostly due to different allocations of tokens in entities. For instance, in *"Oxford Radcliffe Hospitals NHS Trust, Department of Otolaryngology - Head and Neck Surgery, Level LG1, West Wing, John Radcliffe Hospital, Oxford, UK, OX3 9DU."*, ChatGPT finds $\underbrace{\text{Department of Otolaryngology - Head and Neck Surgery}}_{\text{ORG}}$ (among others), whereas Flair finds: $\underbrace{\text{Department of Otolaryngology}}_{\text{ORG}}$ and $\underbrace{\text{Head and Neck Surgery}}_{\text{ORG}}$, two entities instead of the correct single one.

NE type disagreements between extractors The most frequent class of such disagreements (Tab. 5) are Flair Locations considered Organizations by ChatGPT. In these cases, ChatGPT is right: the entity is really an Organization. Some of them include a Location element, e.g., "Middle East Technical University", "The University of Tokyo", "Taipei Medical University", "McGill University", which may have confused Flair, but others do not, e.g., "INRA", "LIFE Center", "Joint Orthopaedic Centre". Conversely, Flair Organizations which ChatGPT considers Locations include: "Lille", "Viet Nam", "Rua de Universidade" (ChatGPT is right; this is a majority of cases), and a handful of cases where Flair is right, e.g., "Ospedale di Busto Arsizio", "Intensive Care Unit". 90% of the Flair Locations and Flair Organizations which GPT finds to be Person entities are initials, such as "A.R.A", "R.H.S", or acronyms such as "M.D" and "PhD". The latter are author academic titles; both extractors are wrong here. The former correspond to authors' initials, found in the paper metadata (conflict of interest statements). ChatGPT is right on "Chiharu Uno" and "L. Giampiero Mazzaglia" (these are people indeed); it also made a mistake on "Korean Firefighters", which should rather be an Organization.

Overall, we find ChatGPT leads to better-quality results, and should be preferred whenever one can afford the budget.

6.3. Path enumeration

For each dataset and pair of entity types, Tab. 6 and 7 report the number of paths of each directionality (Sec. 4.1), the minimum and maximum length L_p of each path, and the minimum and maximum data path support (number of results when evaluated on the data), this is denoted S_p. For the PubMed (XML) and YelpBusiness (JSON) datasets, we obtained only shared-root paths: this is because of the tree structure of these datasets, where text values (leaves) are only connected by going through a common ancestor node. The JSON datasets are more irregular, leading to more paths. Both the RDF Nasa and relational IMDB datasets also contain general-directionality paths.

In almost every case, a few collection paths had 0 support, due to approximations introduced by dataset summarization (Sec. 4). For the IMDB dataset only, we show results only for the 20 top-ranked paths instead of all the enumerated ones (there were 3,620). Among these, all (τ_L, τ_L) paths turned out to have no support at all in the data, despite being present in the collection graph.

These results show that numerous interesting entity paths exist in our datasets, of significant length (up to 9), and some with high support, bringing the need for an efficient evaluation method.

6.4. Efficiency of path evaluation

We now study the efficiency of materializing data paths over the graph (Tab. 8 and Tab. 9). We did not include the IMDB dataset for the same reason as described in Sec. 6.3. However, we provide efficiency results for the top-ranked paths found in the IMDB dataset in Sec. 6.6.

In Tab. 8 and Tab. 9, for each dataset and entity type pair, T_0 is the time to evaluate the corresponding path queries without the view-based optimization of Sec. 5.2 and 5.3, referred to as **VBO** from now on. $|Q_{TO}|$ is the number of queries whose execution we stopped (time-out of 30s) without VBO. $|Q_{NV}|$ is the number of queries for which Algo. 1 did not recommend a view. T_R is the time to evaluate the rewritten queries on the data graph, while $T_{Q_{NV}}$ is the time to evaluate the non-rewritten queries Q_{NV}; $T = T_R + T_{Q_{NV}}$ is the (total) execution time to evaluate queries using VBO. Finally, $s = T_0/T$ is the speed-up due to VBO. We do not report times to materialize views because they were all very short (less than 0.01s). All times are in seconds.

The evaluation time T_0 without VBO ranges from 30s to 1,300s; these path queries require 2 to 9 joins, on graphs of up to more than 200,000 edges (Tab. 2). $|Q_{NV}|$, the number of queries that could not make use of any views, is rather small, which is good. The number of candidate views, respectively, materialized views

(τ_1, τ_2)	N_{root}	$\min(L_p)$	$\max(L_p)$	$\min(S_p)$	$\max(S_p)$
(τ_P, τ_O)	21	5	8	0	13,988
(τ_P, τ_L)	21	5	8	0	15,181
(τ_L, τ_O)	21	5	8	0	5,054
(τ_P, τ_P)	21	5	8	0	389
(τ_L, τ_L)	21	5	8	0	1,214
(τ_O, τ_O)	21	5	8	3	3,090

(τ_1, τ_2)	N_{root}	N_{gen}	$\min(L_p)$	$\max(L_p)$	$\min(S_p)$	$\max(S_p)$
(τ_P, τ_O)	99	1	5	9	0	629
(τ_P, τ_L)	95	5	5	9	0	137
(τ_L, τ_O)	97	3	5	9	0	603
(τ_P, τ_P)	97	3	5	9	0	89
(τ_L, τ_L)	97	3	5	9	0	3,050
(τ_O, τ_O)	97	3	5	9	0	8,960

(τ_1, τ_2)	N_{root}	N_{gen}	$\min(L_p)$	$\max(L_p)$	$\min(S_p)$	$\max(S_p)$
(τ_P, τ_O)	18	5,182	5	9	0	35,963
(τ_P, τ_L)	18	5,182	5	9	0	59,283
(τ_L, τ_O)	25	5,234	5	9	0	9,192
(τ_P, τ_P)	4	3,567	5	9	0	4,729
(τ_L, τ_L)	9	3,620	5	9	0	0
(τ_O, τ_O)	9	3,620	5	9	0	4,263

Table 6: Entity paths in PubMed (top), Nasa (middle) and IMDB (bottom).

(τ_1, τ_2)	N_{root}	$\min(L_p)$	$\max(L_p)$	$\min(S_p)$	$\max(S_p)$
(τ_P, τ_O)	41	5	7	0	651
(τ_P, τ_L)	33	5	7	0	193
(τ_L, τ_O)	21	5	5	0	1,412
(τ_P, τ_P)	28	5	7	0	35
(τ_L, τ_L)	15	5	5	2	158
(τ_O, τ_O)	21	5	5	0	1,232

(τ_1, τ_2)	N_{root}	$\min(L_p)$	$\max(L_p)$	$\min(S_p)$	$\max(S_p)$
(τ_P, τ_O)	48	5	7	0	2,593
(τ_P, τ_L)	39	5	7	0	760
(τ_L, τ_O)	39	5	5	0	258
(τ_P, τ_P)	36	5	7	0	207
(τ_L, τ_L)	15	5	5	0	674
(τ_O, τ_O)	21	5	5	0	4,889

Table 7: Entity paths in the YelpBusiness (top) and YelpBusiness4 (bottom) datasets.

(τ_1, τ_2)	T_0	$\lvert Q_{TO}\rvert$	$\lvert Q_{NV}\rvert$	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{M}\rvert$	T_R	$T_{Q_{NV}}$	$T{=}T_R{+}T_{Q_{NV}}$	$s{=}T_0/T$
PubMed									
(τ_P, τ_O)	250.36	5	1	16	5	3.78	0.32	4.10	61×
(τ_P, τ_L)	37.29	0	1	16	5	19.06	0.32	19.38	2×
(τ_L, τ_O)	151.29	2	2	16	5	11.88	8.59	20.47	7×
(τ_P, τ_P)	152.59	3	1	16	5	44.19	0.08	44.27	3×
(τ_L, τ_L)	169.64	2	1	16	5	71.32	0.31	71.63	2×
(τ_O, τ_O)	317.92	5	1	16	5	22.99	0.25	23.24	13×
Nasa									
(τ_P, τ_O)	195.47	1	0	80	10	54.14	N/A	54.14	3×
(τ_P, τ_L)	254.26	3	0	68	10	44.57	N/A	44.57	5×
(τ_L, τ_O)	1073.55	32	0	77	9	131.58	N/A	131.58	8×
(τ_P, τ_P)	278.95	4	0	76	10	92.01	N/A	92.01	3×
(τ_L, τ_L)	1103.48	30	0	77	9	101.35	N/A	101.35	10×
(τ_O, τ_O)	1318.78	37	0	77	9	247.43	N/A	247.43	5×

Table 8: Data paths evaluation on the PubMed and Nasa datasets.

(τ_1, τ_2)	T_0	$\lvert Q_{TO}\rvert$	$\lvert Q_{NV}\rvert$	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{M}\rvert$	T_R	$T_{Q_{NV}}$	$T{=}T_R{+}T_{Q_{NV}}$	$s{=}T_0/T$
YelpBusiness									
(τ_P, τ_O)	205.95	2	0	22	6	4.20	N/A	4.20	49×
(τ_P, τ_L)	410.87	7	1	19	5	40.87	1.27	42.12	9×
(τ_L, τ_O)	239.90	0	1	20	10	1.15	0.6	1.75	137×
(τ_P, τ_P)	466.58	9	2	23	5	15.33	12.02	27.35	17×
(τ_L, τ_L)	450.00	15	1	8	4	9.89	< 0.01	9.89	45×
(τ_O, τ_O)	334.22	4	1	10	5	2.83	< 0.01	2.83	118×
YelpBusiness4									
(τ_P, τ_O)	804.70	26	0	23	6	62.52	N/A	62.52	12×
(τ_P, τ_L)	454.19	10	1	20	5	92.50	< 0.01	92.50	5×
(τ_L, τ_O)	242.57	5	1	10	5	62.74	6.61	69.35	3×
(τ_P, τ_P)	317.00	7	1	27	7	14.35	1.08	15.43	20×
(τ_L, τ_L)	395.49	10	1	8	4	2.62	18.15	20.77	19×
(τ_O, τ_O)	347.23	8	1	10	5	42.93	2.34	45.27	7×

Table 9: Data path evaluation on the YelpBusiness datasets.

depend on the complexity of the dataset, and thus on the complexity of the paths. The total path evaluation time T is reasonable. Finally, the VBO speed-up is at least 2× and at most 137×, showing that our view-based algorithm allows to evaluate path queries much more efficiently.

6.5. Path reliability and ranking

We now study how reliability (Sec. 4.3) and force (Sec. 4.4) vary across paths. Tab. 10 illustrates reliability of enumerated paths on our datasets, loaded with the ChatGPT-based NE extractor (Sec. 3). We excluded YelpBusiness4 and IMDB in order to limit ChatGPT expenses. For each dataset and a pair of NE types connected by at least a path in a collection graph, we show: min p_{rel}, the minimal path reliability among all the paths connecting NEs of these types; max p_{rel}, the maximal path reliability for the same paths; p^{20}_{rel}, the reliability of the 20th ranked path (20 is the default number of paths shown to the user for each pair of entity types); $|\mathcal{P}|$, the number of enumerated paths; $|\mathcal{P}'|$, which is either 20 if there are at least 20 paths, or $|\mathcal{P}|$ otherwise. We also show the ratio R between $|\mathcal{P}'|$ and $|\mathcal{P}|$. Path reliability values span over the whole $(0, 1]$ interval, e.g., 1.000 or 0.9997 in the YelpBusiness dataset, or 0.9774 in PubMed, to 0.0002 for some in YelpBusiness. Thus, reliability gives a strong signal for ranking paths. Finally, we show $|\mathcal{P}_0|$, the number of enumerated paths leading to no data path, thus having a support of 0. In our experiments, both complex collection graphs (not trees, and possibly with cycles) and low max p_{rel} tend to augment the number of zero-support paths.

Tab. 11 illustrates some paths between τ_P and τ_O, *ordered by reliability, then force, then length*, in the PubMed dataset; we picked this path group since it is the largest (52 initially, recall Tab. 10). It features the first six and the last two paths among the top-ranked paths, as well as p^{21} and p^{22}. For each of them, Tab. 11 shows the path reliability, its force, length and support. We show the reliability of each extraction edge as an index, and similarly, the force of each non-extraction edge, when it is smaller than 1, e.g., the edge connecting the XML element ⟨AuthorList⟩ to all its ⟨Author⟩ children has a force of 0.02.

p^1 connects authors with the organizations to which they are affiliated. This path is ranked first because it is both reliable (p_{rel}=0.914) and strong ($F(p)$=1). Indeed, only people have been found in author names and most of the affiliations are identified as Organization entities (recall Tab. 4). Moreover, all collection edges are strong because each author has only one affiliation in the PubMed dataset. The path also has high support (13,988 data paths). It expresses employment relationships between people and organizations; such connections are clearly very significant, and our ranking captures well this importance.

p^2 connects authors to the organizations found in titles of the journals where their articles appear. The path reliability is around 0.4 (much lower than for p^1). This is because organizations do not appear in all journal titles. However, important journals do carry organization names in their titles, and publishing in such a journal does establish a connection between the author and the organization (the author

(τ_1, τ_2)	min p_{rel}	max p_{rel}	p^{20}_{rel}	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{P}'\rvert$	$R{=}\frac{\lvert\mathcal{P}'\rvert}{\lvert\mathcal{P}\rvert}$	$\lvert\mathcal{P}_0\rvert$
PubMed							
(τ_P, τ_O)	0.0150	0.9142	0.0409	52	20	38.45%	6
(τ_P, τ_L)	0.0150	0.9107	0.0150	30	20	66.66%	8
(τ_L, τ_O)	0.0150	0.9107	0.0232	34	20	58.82%	6
(τ_P, τ_P)	0.0150	0.9774	0.0150	24	20	83.33%	11
(τ_O, τ_O)	0.0150	0.4158	0.0232	31	20	64.51%	6
(τ_L, τ_L)	0.0150	0.0954	0.0150	20	20	100.00%	9
Nasa							
(τ_P, τ_O)	0.0014	0.0645	0.0178	191	20	10.47%	15
(τ_P, τ_L)	0.0014	0.0645	0.0077	142	20	14.08%	16
(τ_L, τ_O)	0.0014	0.1016	0.0077	115	20	17.39%	10
(τ_P, τ_P)	0.0014	0.0232	0.0077	110	20	18.18%	15
(τ_O, τ_O)	0.0014	0.0581	0.0077	92	20	21.73%	15
(τ_L, τ_L)	0.0014	0.3790	0.0077	67	20	29.85%	12
YelpBusiness							
(τ_L, τ_O)	0.0002	0.9997	0.0002	8	8	100.00%	8
(τ_L, τ_L)	0.0002	1.0000	0.0002	11	11	100.00%	2

Table 10: Numbers of paths, reliability information and zero-support paths in our datasets.

p^{id}	p_{rel}	$F(p)$	$L(p)$	$S(p)$	Path p connecting a Person (τ_P) entity with an Organization (τ_O) entity
p^1	0.914	1.00	6	13,988	$\tau_P \leftarrow_{1.0}$ Name#val $\leftarrow$ Name $\leftarrow$ Author $\rightarrow$ Affiliation $\rightarrow$ Affiliation#val $\rightarrow_{0.914} \tau_O$
p^2	0.416	0.02	8	3,044	$\tau_P \leftarrow_{1.0}$ Name#val $\leftarrow$ Name $\leftarrow$ Author $\leftarrow_{0.02}$ AuthorList $\leftarrow$ PubmedArticle $\rightarrow$ JournalTitle $\rightarrow$ JournalTitle#val $\rightarrow_{0.416} \tau_O$
p^3	0.098	1.00	6	65	$\tau_P \leftarrow_{0.098}$ CoiStatement#val $\leftarrow$ CoiStatement $\leftarrow$ PubmedArticle $\rightarrow$ JournalTitle $\rightarrow$ JournalTitle#val $\rightarrow_{0.416} \tau_O$
p^4	0.098	0.02	8	1,748	$\tau_P \leftarrow_{0.098}$ CoiStatement#val $\leftarrow$ CoiStatement $\leftarrow$ PubmedArticle $\rightarrow$ AuthorList $\rightarrow_{0.02}$ Author $\rightarrow$ Affiliation $\rightarrow$ Affiliation#val $\rightarrow_{0.914} \tau_O$
p^5	0.090	1.00	2	1,173	$\tau_P \leftarrow_{0.098}$ CoiStatement#val $\rightarrow_{0.090} \tau_O$
p^6	0.090	0.02	8	2,327	$\tau_P \leftarrow_{1.0}$ Name#val $\leftarrow$ Name $\leftarrow$ Author $\leftarrow_{0.02}$ AuthorList $\leftarrow$ PubMedArticle $\rightarrow$ CoiStatement $\rightarrow$ CoiStatement#val $\rightarrow_{0.090} \tau_O$
...	...	...	...	...	...
p^{19}	0.042	0.02	8	488	$\tau_P \leftarrow_{0.042}$ ArticleTitle#val $\leftarrow$ ArticleTitle $\leftarrow$ PubmedArticle $\rightarrow$ AuthorList $\rightarrow_{0.02}$ Author $\rightarrow$ Affiliation $\rightarrow$ Affiliation#val $\rightarrow_{0.914} \tau_O$
p^{20}	0.041	1.00	2	15	$\tau_P \leftarrow_{0.042}$ ArticleTitle#val $\rightarrow_{0.041} \tau_O$
p^{21}	0.041	1.00	6	5	$\tau_P \leftarrow_{0.098}$ CoiStatement#val $\leftarrow$ CoiStatement $\leftarrow$ PubmedArticle $\rightarrow$ ArticleTitle $\rightarrow$ ArticleTitle#val $\rightarrow_{0.041} \tau_O$
p^{22}	0.041	0.02	8	349	$\tau_P \leftarrow_{1.0}$ Name#val $\leftarrow$ Name $\leftarrow$ Author $\leftarrow_{0.02}$ AuthorList $\leftarrow$ PubmedArticle $\rightarrow$ ArticleTitle $\rightarrow$ ArticleTitle#val $\rightarrow_{0.041} \tau_O$

Table 11: Some of the top-reliability (τ_P, τ_O) paths in the PubMed dataset, at ranks: 1 to 6 and 19 and 20 (above the double line), respectively, 21 and 22 (below the double line), out of 52 paths.

knows the organization exists, agrees or is even proud to be associated to it by publishing, and probably would regret if the organization were to close or lose reputation). This path has a respectable support (3,044 data paths); intuitively, it is also clearly interesting.

The next two paths, p^3 and p^4, connect people found in conflict of interest (COI) statements, to Organizations found in the titles of the journals in which those papers appear, respectively in the authors affiliations. p^3 ranks higher because there is only one way to connect a COI statement to a journal title: the article itself, thus the force is 1. On the contrary, in p^4, a COI statement can be associated to any of the authors of a paper; a paper has up to 50 authors in the PubMed dataset. This considerably lowers the force of p^4, to only 0.02. p^3 has a much lower support, because few journal titles contain organizations.

p^5 connects Person and Organization NEs found in the same COI statement value node, thus is a strong path. p^6 connects any paper author (Person NE) to Organizations mentioned in the COI statements of those papers, so this path is much weaker (force of 0.02). Both paths have a p_{rel} of 0.090, so the force is used to rank them. They also both have high support, and may be highly interesting when investigating conflicts of interests between research labs and companies.

p^{19} connects Person NEs found in article titles to Organization NEs found in Affiliations of the paper authors. It has a reasonable support (488 data paths) even if its reliability has decreased by 2 since p^6, and it is weak due to the possibly high number of authors for a given paper.

p^{20}, p^{21} and p^{22} connect respectively: (*i*) Person and Organization NEs found in article titles; (*ii*) people extracted from COI statements to the Organization NEs found in titles of papers having that COI statement; (*iii*) author names (Person NEs) to the Organization NEs found in their articles' titles. The three paths all have a path reliability of 0.041 due to the extraction edge for Organization NEs in article titles. p^{20} and p^{21} are both strong; p^{20} is ranked first since it is shorter. p^{22} is weaker, because it may connect any of the 50 authors of an article to the article title. These three paths show how to connect Organizations found in articles titles, such as "Evaluation of the [...] Opioid Prescribing Risk Evaluation and Mitigation Strategy Program by the $\underbrace{\text{US Food and Drug Administration}}_{\text{ORG}}$: A Review", to paper authors. They lead to dozens, up to hundreds, of very interesting connections between people and companies, which may be further investigated, e.g., within a journalism project.

The least reliable collection extraction edges found in PubMed include: 6 Locations extracted from 399 CoiStatement values (leading to a reliability of 0.015)

and 86 people names in 5,712 Affiliations (reliability also equal to 0.015). They were found in strings such as "$\underbrace{\text{James J. Peters}}_{PERS}$ VA Medical Center, 130 West Kingsbridge Road, Bronx, NY 10468, USA.". All paths traversing this edge are at the very bottom of the ranked list of paths. Analysis of path reliability in the other datasets lead to similar findings. Overall, Tab. 11 shows that ranking paths based on their reliability, force, and length if a second tie break is needed, favors paths that (*i*) are less likely to contain NE extraction errors, and (*ii*) do not dilute information along the path.

6.6. Evaluation of the top-ranked paths

Tab. 12 studies the benefit of the MQO approach (Algorithm 1) on the top-ranked paths. For each dataset, we show the same information as presented in Tab. 8 and 9 (Sec. 6.4). PubMed, Nasa and YelpBusiness have been ingested with the ChatGPT-based NE extractor, while we used Flair for IMDB due to costs. All the evaluation times, notably T_0 when directly evaluating paths, and T when applying our MQO, are given in seconds. The speed-up (rightmost column) shows that for 12 out of 20 path groups, MQO achieves its goal, reducing evaluation times by up to 10× or 20×. We also notice two path groups whose evaluation is slowed by MQO, by a factor of 2×, respectively, 5×. This is due to its heuristic choices and relatively simple cost model. The gains are slightly less than those in Tab. 9; this may be because there were in general more paths in that table, leading to more sharing opportunities. Still, we find the MQO gains are generally robust and significant, confirming its interest.

6.7. Experiment conclusion

Our experiments lead to the following observations. First, the ChatGPT entity extractor we developed improved over our prior work Barret et al. (2023b) by (*i*) better recognition of Locations, with all geographical levels (street up to country); (*ii*) significantly better recognition of Organizations, avoiding false positives and finding good entity labels; (*iii*) modest improvements in the quality of extracting people names; and (*iv*) overall, better support of many languages (Sec. 6.2). All these advantages can be attributed to its large training corpus, and recommend it whenever extraction time is not crucial, and financial costs can be afforded. Second, many named entity paths exists, in the datasets we considered (Sec. 6.3). Third, our path evaluation algorithm is very effective in reducing the time to evaluate path sets (Sec. 6.4). Fourth, our novel ranking based on reliability and force, a novelty we introduced since Barret et al. (2023b), downgrades many

(τ_1, τ_2)	T_0	$\lvert Q_{TO}\rvert$	$\lvert Q_{NV}\rvert$	$\lvert \mathcal{V}\rvert$	$\lvert \mathcal{M}\rvert$	T_R	$T_{Q_{NV}}$	$T{=}T_R{+}T_{Q_{NV}}$	$s{=}T_0/T$
PubMed									
(τ_P, τ_O)	6.29	0	1	32	4	1.80	0.04	1.84	3.4×
(τ_P, τ_L)	38.73	1	1	26	4	1.57	0.02	1.59	24.3×
(τ_L, τ_O)	68.80	2	1	24	4	4.04	0.02	4.06	16.9×
(τ_P, τ_P)	2.21	0	4	26	5	9.08	0.04	9.12	0.2×
(τ_O, τ_O)	20.20	0	1	33	4	18.65	0.01	18.66	1.1×
(τ_L, τ_L)	23.57	0	7	16	2	1.38	0.01	1.39	16.9×
Nasa									
(τ_P, τ_O)	11.30	0	1	46	2	10.04	8.82	18.86	0.5×
(τ_P, τ_L)	29.77	0	1	41	4	2.37	0.03	2.40	12.4×
(τ_L, τ_O)	36.57	1	1	29	4	4.71	0.03	4.74	7.7×
(τ_P, τ_P)	3.13	0	1	35	2	2.46	0.01	2.47	1.2×
(τ_O, τ_O)	32.66	1	3	35	4	2.89	0.04	2.93	11.1×
(τ_L, τ_L)	36.13	0	6	19	2	1.62	0.09	1.71	21.1×
YelpBusiness									
(τ_L, τ_O)	38.86	1	0	7	1	3.62	0	3.62	10.7×
(τ_L, τ_L)	131.98	4	2	6	3	45.74	0.15	45.89	2.8×
IMDB									
(τ_P, τ_O)	550.96	15	0	32	7	59.86	N/A	59.86	9.2×
(τ_P, τ_L)	453.72	8	0	38	6	140.61	N/A	140.61	3.2×
(τ_L, τ_O)	486.24	12	0	34	4	54.73	N/A	54.73	8.9×
(τ_P, τ_P)	314.28	8	10	10	4	26.04	37.35	63.39	4.9×
(τ_L, τ_L)	232.38	6	11	14	2	20.17	30.13	50.30	4.6×
(τ_O, τ_O)	242.09	4	11	12	3	21.44	29.18	50.62	4.8×

Table 12: Data path evaluation on the top-20 enumerated paths, sorted by reliability, then force, in the PubMed, Nasa, YelpBusiness and IMDB datasets.

meaningless paths, while preserving significant ones (Sec. 6.5). Our joint path materialization technique is effective also on the top-ranked paths (Sec. 6.6).

7. Related work

We discuss below how our work is placed in the areas of graph exploration (Sec. 7.1), respectively, materialized view recommendation (Sec. 7.2).

7.1. Graph database exploration

To extract information from graph databases, users can rely, first, on *query languages*, such as SPARQL 1.1 for RDF graphs, or GPML Deutsch et al. (2022)

for property graphs. However, non-technical users find this too hard. The notion of nodes connected by paths is instead quite intuitive, thus it could be tempting to ask a graph query processor to return paths between certain types of entities. However, GPML and SPARQL allow finding pairs of nodes reachable from one another, but not the paths connecting them. In contrast, in an application domain, the labels of edges along the paths encapsulate useful information which helps understand what the connection is about.

Other graph exploration works extend SPARQL to constrain the labels on the paths connecting nodes Aebeloe et al. (2018), or help users specify their information needs by example, while the system formulates and evaluates the corresponding SPARQL queries, e.g., conjunctive in Lissandrini et al. (2022), and also with aggregation in Lissandrini et al. (2023). Such efforts are complementary; unlike our system, they assume an underlying SPARQL query processor, and are not meant to help users discover data paths. Another related line of work is keyword-based search in graphs Agrawal et al. (2002); Anadiotis et al. (2022); Yang et al. (2021), where users asks for connections between two or more nodes matching specific keywords. This is an alternative graph discovery mode; our recent KBS algorithm Anadiotis et al. (2023) is integrated in the ConnectionStudio tool Barret et al. (2023a), along with Abstra and PathWays.

Abstra introduces a particular graph summarization method Barret et al. (2024), which is a quotient method (based on grouping all nodes in equivalence classes), adapted to several data models. Other graph summarization methods, either quotients, or based on other structural, statistic, or semantic criteria have been surveyed in Cebiric et al. (2019) and Liu et al. (2018).

Starting from Abstra, in Barret et al. (2023c), we introduced the novel problem of efficiently materializing NE paths. The present work improves over these by a better NE extractor based on ChatGPT, and by ranking (as opposed to pruning) paths, reducing user effort.

7.2. Materialized view recommendation

Materialized view recommendation (MVR, in short) is a well-studied problem Mami and Bellahsene (2012). Given a query q, a view v that stores partial results of a costly subquery of q may dramatically decrease the cost of evaluating q. Because production workloads often contain similar queries, there is an obvious interest in identifying views that may serve *several* queries $q_1, q_2, \ldots$. Such views are typically *common subexpressions* of the q_i's, leading to a natural connection with *multi-query optimization* Roy et al. (2000); Roy and Sudarshan

(2009), seeking to simultaneously improve the evaluation performance for a set of queries.

MVR algorithms have been proposed for relational databases, e.g., Ligoudistianos et al. (1999), for RDF databases Le et al. (2012), including methods taking into account implicit data due to ontologies Goasdoué et al. (2011), for XML Katsifodimos et al. (2012), etc. Each view recommendation method explores a certain space of candidate views, typically subexpressions of one or more input queries. The larger this space, the more complex features each view has, the longer will be spent enumerating them, and estimating their performance benefits; in exchange, highly-tuned views may bring very important performance savings, especially if the model used to estimate the cost of view-based evaluation is very accurate. On the contrary, if the space of candidate views is small, it is easily explored, but the best cost savings may be missed. Thus, in a traditional query processing setting, a good approach is to first, collect a workload, then, spend time selecting the best views, and deploy them; users will benefit directly from an efficient execution.

Our setting brings some extra hypotheses to the MVR problem. By design, all our queries are paths, which can be seen as a particular case of those studied in Ligoudistianos et al. (1999); Le et al. (2012); Goasdoué et al. (2011): any sub-expression of a path query (that does not contain a cartesian product) is also a path. Thus, we can afford to only explore a space of path views, unlike these prior works, whose MVR methods are more complex. Second, PathWays builds on top of Abstra Barret et al. (2024) and ConnectionLens Anadiotis et al. (2022), which store the graph within Postgres. As a consequence, path queries, the views, and view-based rewritings are evaluated as SQL queries by Postgres, which optimizes them internally. An advantage of this is that we can leverage Postgres' cost and cardinality estimation, which are quite mature and of good quality. A difficulty raised is that we have to also "guess" the cost of Postgres' view-based query evaluation. Fortunately, Postgres' cost model is quite "textbook", and its choices can be reasonably well predicted, as in, e.g., Bursztyn et al. (2015, 2016). Third, PathWays enumerates paths on request, following users' choice of entity types, maximum path lengths, etc. Thus, we wanted a fast MVR method, therefore we constrained the candidate views to the maximal shared path segments among two or more paths. This is a pragmatic choice that worked well in practice, but it may not preserve optimality, i.e., some subpaths of these maximal shared segments may bring even better performance. We opted against exploring such shorter views, also because the prediction of view-based query evaluation performance over many joins becomes more hazardous.

8. Conclusion and perspectives

In this work, we have studied a new graph data exploration method, aimed at non technical users who are not yet familiar with the data content and structure. Our method consists of showing them paths which connect pairs of named entities, of types that the users select. We enumerate such paths based on a structural summary built by the prior Abstra system Barret et al. (2024). In order to speed up the evaluation of a set of paths, we use a materialized view recommendation method, that uses maximum shared sub-paths as materialized views. Our platform expresses views and rewritings as SQL queries, and evaluates them through Postgres. Improving upon our prior work Barret et al. (2023b), we have devised a more accurate Named Entity extractor using ChatGPT, and devised novel metrics for ranking paths, reliability and force, so that our evaluation effort and the users' attention can be focused on our highest-rank paths.

Numerous continuations of this work can be envisioned. For what concerns the NER task, we can leverage ChatGPT to extract more specific entities, e.g., separate Universities from Research Funding Agencies and Companies, instead of considering them all to be Organizations. We are planning to explore the usage of other large, free models, trying to get close to ChatGPT's accuracy, without its monetary costs. Another area of future work is to combine the path enumeration and evaluation with user-specified natural language search, in order to prioritize the ranking and evaluation of the paths most interesting to them. One may also study entity paths from the perspective of their overlap with multiple Abstra entities, and/or multiple heterogeneous datasets. Finally, we may investigate graph metrics, others than reliability and force, since they may lead to other notions of interestingness among the enumerated paths.

Acknowledgements

We thank the journalists Stéphane Horel (Le Monde) and Camille Pettineo (INA, France), as well as the DataJournos association and the CFI 2023 attendants, for their remarks and useful feedback. This work is partially funded by DIM RFSI PHD 2020-01 and ANR-20-CHIA-0015-01 grants.

References

Abedjan, Z., Golab, L., Naumann, F., Papenbrock, T., 2018. Data Profiling. Synthesis Lectures on Data Management, Morgan & Claypool Publishers.

Aebeloe, C., Setty, V., Montoya, G., Hose, K., 2018. Top-K Diversification for Path Queries in Knowledge Graphs, in: ISWC Workshops.

Agrawal, S., Chaudhuri, S., Das, G., 2002. DBXplorer: A system for keyword-based search over relational databases, in: ICDE.

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., Vollgraf, R., 2019. Flair: An easy-to-use framework for state-of-the-art NLP, in: ACL.

Anadiotis, A.C., Balalau, O., Bouganim, T., Chimienti, F., Galhardas, H., Haddad, M.Y., Horel, S., Manolescu, I., Youssef, Y., 2021. Empowering Investigative Journalism with Graph-based Heterogeneous Data Management. Bulletin of the Technical Committee on Data Engineering URL: https://hal.science/hal-03337650.

Anadiotis, A.C., Manolescu, I., Mohanty, M., 2023. Integrating Connection Search in Graph Queries, in: ICDE.

Anadiotis, A.G., Balalau, O., Conceição, C., Galhardas, H., Haddad, M.Y., Manolescu, I., Merabti, T., You, J., 2022. Graph integration of structured, semistructured and unstructured data for data journalism. Inf. Syst. 104, 101846. URL: https://doi.org/10.1016/j.is.2021.101846, doi:10.1016/J.IS.2021.101846.

Barret, N., Ebel, S., Galizzi, T., Manolescu, I., Mohanty, M., 2023a. User-friendly exploration of highly heterogeneous data lakes, in: CoopIS, Springer. pp. 488–496. URL: https://doi.org/10.1007/978-3-031-46846-9_30, doi:10.1007/978-3-031-46846-9_30.

Barret, N., Gauquier, A., Law, J.J., Manolescu, I., 2023b. Exploring heterogeneous data graphs through their entity paths, in: Advances in Databases and Information Systems, Springer. pp. 163–179. URL: https://doi.org/10.1007/978-3-031-42914-9_12, doi:10.1007/978-3-031-42914-9_12.

Barret, N., Gauquier, A., Law, J.J., Manolescu, I., 2023c. Pathways: Entity-focused exploration of heterogeneous data graphs, in: The Semantic Web: ESWC 2023 Satellite Events, Springer. pp. 91–95. URL: https://doi.org/10.1007/978-3-031-43458-7_17, doi:10.1007/978-3-031-43458-7_17.

Barret, N., Manolescu, I., Upadhyay, P., 2022. Abstra: toward generic abstractions for data of any model (demonstration), in: CIKM.

Barret, N., Manolescu, I., Upadhyay, P., 2024. Computing generic abstractions from application datasets, in: 27th International Conference on Extending Database Technology, OpenProceedings.org. pp. 94–107. URL: https://doi.org/10.48786/edbt.2024.09, doi:10.48786/EDBT.2024.09.

Bursztyn, D., Goasdoué, F., Manolescu, I., 2015. Optimizing reformulation-based query answering in RDF, in: Alonso, G., Geerts, F., Popa, L., Barceló, P., Teubner, J., Ugarte, M., den Bussche, J.V., Paredaens, J. (Eds.), EDBT, OpenProceedings.org. pp. 265–276. URL: https://doi.org/10.5441/002/edbt.2015.24, doi:10.5441/002/EDBT.2015.24.

Bursztyn, D., Goasdoué, F., Manolescu, I., 2016. Teaching an RDBMS about ontological constraints. Proc. VLDB Endow. 9, 1161–1172. URL: http://www.vldb.org/pvldb/vol9/p1161-bursztyn.pdf, doi:10.14778/2994509.2994532.

Cebiric, S., Goasdoué, F., Kondylakis, H., Kotzinos, D., Manolescu, I., Troullinou, G., Zneika, M., 2019. Summarizing semantic graphs: a survey. VLDB J. 28, 295–327. URL: https://doi.org/10.1007/s00778-018-0528-3, doi:10.1007/S00778-018-0528-3.

Coffman, J., Weaver, A.C., 2014. An empirical performance evaluation of relational keyword search techniques. IEEE Trans. Knowl. Data Eng. 26, 30–42. URL: https://doi.org/10.1109/TKDE.2012.228, doi:10.1109/TKDE.2012.228.

Deutsch, A., Francis, N., Green, A., Hare, K., Li, B., Libkin, L., et al., 2022. Graph pattern matching in GQL and SQL/PGQ, in: SIGMOD.

Goasdoué, F., Guzewicz, P., Manolescu, I., 2020. RDF graph summarization for first-sight structure discovery. VLDB J. 29, 1191–1218. URL: https://doi.org/10.1007/s00778-020-00611-y, doi:10.1007/S00778-020-00611-Y.

Goasdoué, F., Karanasos, K., Leblay, J., Manolescu, I., 2011. View selection in semantic web databases. PVLDB 5.

Halevy, A.Y., 2001. Answering queries using views: A survey. VLDB J. 10.

Hristidis, V., Papakonstantinou, Y., Balmin, A., 2003. Keyword proximity search on XML graphs, in: ICDE.

Jiang, L., Naumann, F., 2020. Holistic primary key and foreign key detection. J. Intell. Inf. Syst. 54.

Katsifodimos, A., Manolescu, I., Vassalos, V., 2012. Materialized view selection for XQuery workloads, in: Candan, K.S., Chen, Y., Snodgrass, R.T., Gravano, L., Fuxman, A. (Eds.), ACM SIGMOS, ACM. pp. 565–576. URL: https://doi.org/10.1145/2213836.2213900, doi:10.1145/2213836.2213900.

Le, W., Kementsietsidis, A., Duan, S., et al., 2012. Scalable multi-query optimization for SPARQL, in: ICDE.

Ligoudistianos, S., Sellis, T.K., Theodoratos, D., Vassiliou, Y., 1999. Heuristic algorithms for designing a data warehouse with SPJ views, in: Mohania, M.K., Tjoa, A.M. (Eds.), DaWaK, Springer. pp. 96–105. URL: https://doi.org/10.1007/3-540-48298-9_10, doi:10.1007/3-540-48298-9_10.

Lissandrini, M., Hose, K., Pedersen, T.B., 2023. Example-driven exploratory analytics over knowledge graphs, in: EDBT.

Lissandrini, M., Mottin, D., Hose, K., Pedersen, T.B., 2022. Knowledge graph exploration systems: are we lost?, in: CIDR, www.cidrdb.org.

Liu, Y., Safavi, T., Dighe, A., Koutra, D., 2018. Graph summarization methods and applications: A survey. ACM Comput. Surv. 51, 62:1–62:34. URL: https://doi.org/10.1145/3186727, doi:10.1145/3186727.

Mami, I., Bellahsene, Z., 2012. A survey of view selection methods. SIGMOD Rec. 41, 20–29. URL: https://doi.org/10.1145/2206869.2206874, doi:10.1145/2206869.2206874.

Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D., 2014. The Stanford CoreNLP natural language processing toolkit, in: ACL (demonstrations).

Manolescu, I., Mohanty, M., 2023. Full-power graph querying: State of the art and challenges. Proc. VLDB Endow. 16, 3886–3889. URL: https://www.vldb.org/pvldb/vol16/p3886-mohanty.pdf, doi:10.14778/3611540.3611577.

Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S., 2000. Efficient and extensible algorithms for multi query optimization, in: Chen, W., Naughton, J.F., Bernstein, P.A. (Eds.), SIGMOD, ACM. pp. 249–260. URL: https://doi.org/10.1145/342009.335419, doi:10.1145/342009.335419.

Roy, P., Sudarshan, S., 2009. Multi-query optimization , 1849–1852URL: https://doi.org/10.1007/978-0-387-39940-9_239, doi:10.1007/978-0-387-39940-9_239.

Yang, J., Yao, W., Zhang, W., 2021. Keyword search on large graphs: A survey. Data Sci. Eng. 6.