Graph Detective: A User Interface for Intuitive Graph Exploration Through Visualized Queries

Dominik Opitz
German Aerospace Center (DLR)
Sankt Augustin, Germany
dominik.opitz@dlr.de

Andreas Hamm
German Aerospace Center (DLR)
Sankt Augustin, Germany
andreas.hamm@dlr.de

Roxanne El Baff
German Aerospace Center (DLR)
Oberpfaffenhofen, Germany
roxanne.elbaff@dlr.de

Jasper Korte
German Aerospace Center (DLR)
Sankt Augustin, Germany
jasper.korte@dlr.de

Tobias Hecking
German Aerospace Center (DLR)
Sankt Augustin, Germany
tobias.hecking@dlr.de

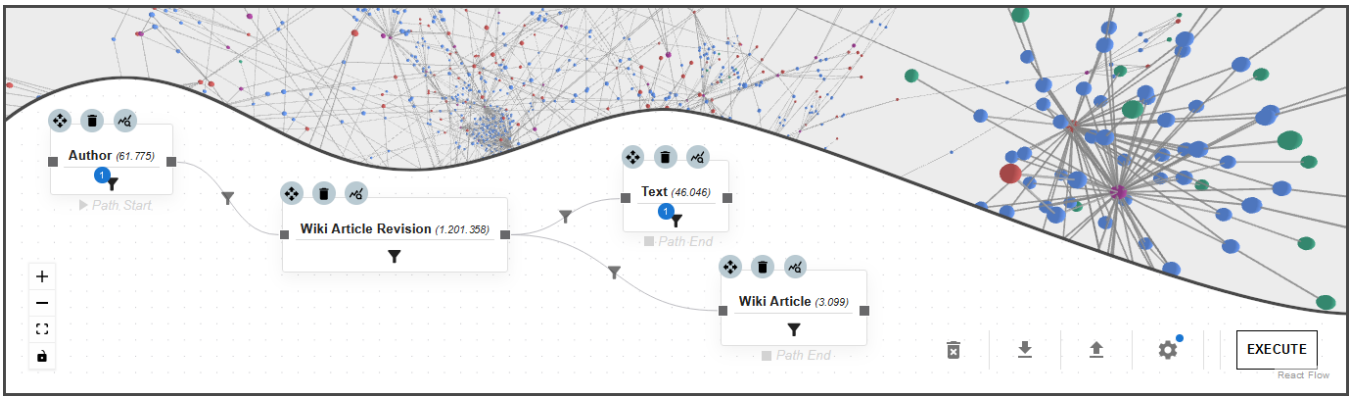


Figure 1: Graph Detective: Modeling graph queries with visual elements and rendering query results in interactive views.

ABSTRACT

Graph databases are used across several domains due to the intuitive structure of graphs. They are well-suited for storing document collections together with their interlinkages through metadata and annotations. Yet, querying such graphs requires database experts' involvement for query formulation, reducing accessibility to non-experts. To address this issue, we present *Graph Detective*, a web interface that provides an intuitive entry point for graph data exploration, where users can create queries visually with little effort, eliminating the need for expertise in query writing. After processing, the resulting query output (a graph) is then rendered in an interactive 3D visualization. This visualization allows the analysis of structural traits of the resulting graph data, exploiting the documents and metadata interlinkage. Our user evaluation revealed that even individuals inexperienced with graph databases or graph data, in general, could satisfactorily access the graph data through our interface. Furthermore, experienced participants commented that our interface was more efficient than writing explicit queries in graph database query language. Interested users can find the code openly accessible on GitHub[1].

DocEng '24, August 20–23, 2024, San Jose, CA, USA

ACM ISBN 979-8-4007-1169-5/24/08
https://doi.org/10.1145/3685650.3685660

CCS CONCEPTS

• **Information systems** → **Search interfaces**; **Query representation**; • **Human-centered computing** → *Visualization*.

KEYWORDS

Graph Database, Query Generation, Graph Retrieval Interface, Web Application, Graph Data, Visual Querying

ACM Reference Format:
Dominik Opitz, Andreas Hamm, Roxanne El Baff, Jasper Korte, and Tobias Hecking. 2024. Graph Detective: A User Interface for Intuitive Graph Exploration Through Visualized Queries. In *ACM Symposium on Document Engineering 2024 (DocEng '24), August 20–23, 2024, San Jose, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3685650.3685660

1 INTRODUCTION

In many domains, graphs constitute a useful and intuitive abstraction, where edges (a.k.a. relations) capture different, potentially complex relations between the entities of a domain [1, 6, 7]. Graphs are stored in graph databases, which excel in handling interconnected data.

[1] https://github.com/DLR-SC/GraphDetective

Graphs represent knowledge valuably based on the conjunction of content, metadata, and extracted annotations in large document collections. Resulting graphs have been used in various domains such as health, engineering, and social media [22] for several downstream tasks such as relation and entity extraction [31, 27], question answering [11], and semantic search [14]. However, before tackling these downstream tasks, different domain-dependent users with no technical background need to inspect and visually analyze the raw graph that is usually stored on machines, requiring machine-compatible access, typically accessed through querying languages such as *Neo4j*[2], *ArangoDB*[3] with their querying language *Cypher* and *ArangoDB Query Language (AQL)* respectively. The lack of proficiency in these querying languages forces many data consumers (e.g., decision-makers) to rely on database experts to retrieve the data, often leading to delays and hindering exploratory data access.

To address this issue, we present a web application with a user interface for 3D graph data visualization: *Graph Detective*. As a core contribution, *Graph Detective* provides an interface for no-code document retrieval from property graphs. The interface overcomes the need to formulate complicated database queries in the form of text. Instead, users model queries visually on a canvas with click-and-drag features. This visual query is then converted to actual database queries using a template-based algorithm. To enable exploration of the graph structures, parts of the interface offer interactive visualizations of the graph result. Our implementation uses ArangoDB as the underlying graph database, but the concept can be applied to other graph databases as well. To showcase the usefulness of *Graph Detective*, we chose use cases outlining document analysis on a linguistically annotated graph.

The interface is logically divided into an **input area**, a **display area** and an **ontology area**. In the input area, the user graphically specifies the entities and relations they want to retrieve from the database. To better understand the graph schema, users can view the graph schema of the underlying graph displayed in the ontology area. Afterward, the system converts the input to a database query and executes it on the database. The result is returned to the user within the output area.

In order to verify the effectiveness of our interface, we conducted a human evaluation. It revealed that participants, both with and without prior knowledge in the domain, can research specific graph information using the interface.

Our main contributions are threefold:

- We introduce a visual query builder that eliminates the need to write complicated database queries using traditional query languages, enabling quick and easy insights for users with different backgrounds;
- We provide a three-dimensional visual representation of the queried graph data in an interactive canvas, giving the user an intuitive understanding of the graph data and enabling a quick understanding of its structure;
- We conducted a user study showing that users with varying prior technical knowledge can retrieve specific graph information using the interface. Through the evaluation, we identify enhancements to improve effectiveness of the interface for users.

2 RELATED WORK

In addressing the challenge faced by users unfamiliar with graph databases, several works explore methods to bridge the gap between a researcher's intent and the execution of a graph query, which includes visual programming tools for creating textual queries through visual interfaces and sequence-to-sequence translation for translating natural language questions into database queries.

Visual Programming

Visual Programming (VP) enables users to create programs through graphical elements rather than textual programming code, which reduces the barrier to entry for end users who are not trained software professionals [16]. Kuhail et al. further describe and classify VP-based tools into four categories: **Form-based** tools allow the placement of graphical components onto a form to construct a functional user interface [15, 24]. **Diagram-based** tools let the user create diagrams by piecing different visual components together where data flows and algorithms [25] are created efficiently. Our proposed *Graph Detective* belongs to this category. Existing related tools often have other limitations. Some are restricted to tabular data [5] or numerical data [13]. Others are closed-source [9] or lack the ability to define visual graph queries, focusing solely on exploration [4]. Our tool distinguishes itself by serving as a property graph database query visualization tool, emphasizing the unique capacity to visually define complex graph queries in ArangoDB. While inspired by existing systems, our focus addresses the specific need for interactive query formulation in property graph databases. **Block-based** tools construct larger structures or workflows by connecting pre-defined visual blocks that seamlessly fit together, much like assembling a jigsaw puzzle [8, 23]. **Icon-based** tools construct a program by connecting data flow icons, where each icon is a service or action [12].

Neural Query Generation

In addition to constructing queries visually, alternative methods involve querying databases based on natural language, allowing users to articulate their information needs more intuitively. Such approaches often adopt neural sequence-to-sequence learning [2, 28] and Large Language Models (LLMs) such as ChatGPT [4] to directly generate a query. However, due to their black-box nature and hallucination issues [29], LLMs still struggle when addressing ambiguous or contextually complex queries [17]. They can not assure the generation of syntactically valid queries, arguably a very important requirement to ensure failure-free execution [18].

3 SYSTEM OVERVIEW

Graph Detective has three distinct components: A canvas area to model visual queries (Figure 2, A), an ontology of the graph database that the interface is connected to (Figure 3) and an interactive graph explorer for visualizing query results (Figure 4). This section

[2]https://neo4j.com
[3]https://arangodb.com
[4]https://openai.com/chatgpt

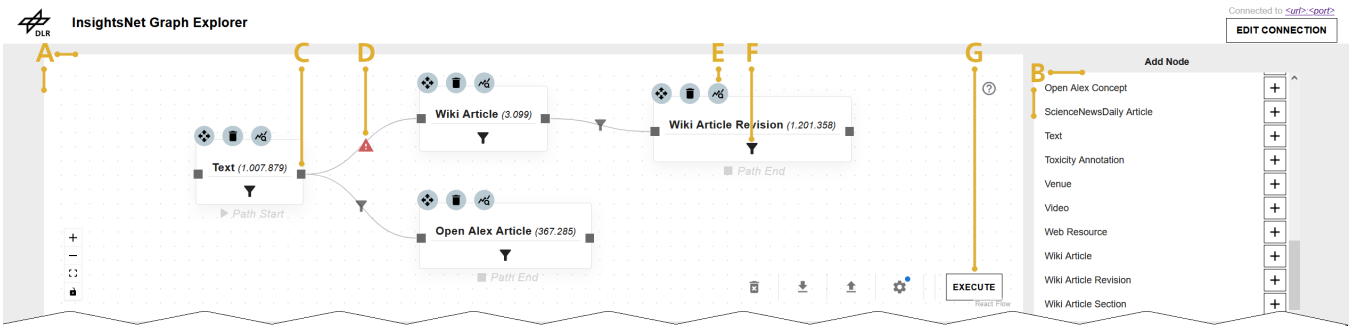


Figure 2: Main Component: Canvas area where a user designs visual database queries.

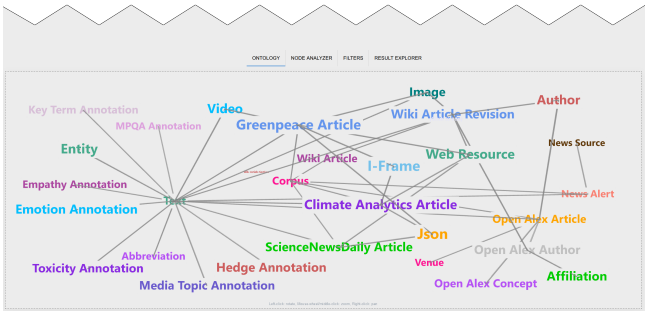


Figure 3: Ontology Graph: An interactive 3D visualization of the metagraph of the graph database that the interface is connected to.

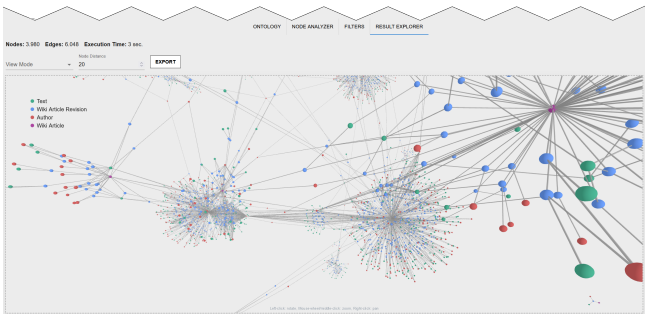


Figure 4: Main Component: Canvas area where a user designs visual database queries.

describes each component (Section 3.1, 3.2 and 3.3) in more detail. However, first, we introduce the components of an ArangoDB graph:

Vertex. In ArangoDB, a vertex is an entity stored as a record, called *document* with several properties. A graph database can have several entity types, such as PERSON with properties (e.g., name, age) or ORGANIZATION (e.g., name, establishment data). A *vertex collection* contains all the records of the same entity type (e.g., PERSON).

Edge. Each edge in a graph represents a relation among two vertices (e.g., PERSON P works at ORGANIZATION O), which is also stored as a record with properties (e.g., the date of joining the organization), where these records are stored in containers called *edge collections*.

Graph. An ArangoDB graph is the network of all vertex and edge records. A graph database query aims to filter a graph by selecting certain vertex/edge types and setting conditions on vertex/edge properties to answer a specific question.

3.1 Canvas Area for Query Modeling

This component empowers users to create a graphical structure of vertex and edge collections (referred to as nodes and edges) on a two-dimensional canvas (Figure 2, A) - it visually represents a database query to be executed. Users can add nodes to the visual database query from an adjacent list of available entity types (Figure 2, B). Once a user drops a vertex onto the canvas, they can access three functionalities for it:

(1) **Linking Vertex Types** A user can drag and drop several nodes and then link them via edges (Figure 2, C). If the linkage does not exist in the graph schema, a warning is displayed (Figure 2, D), indicating an invalid edge. The query is not executable with the presence of invalid edges, as it would result in a semantically invalid query.
(2) **Document Viewer** To understand the individual documents belonging to a vertex or edge collection, a user can scan each node type individually in an ad-hoc fashion by clicking on the corresponding icon in the node (Figure 2, E). This lists all documents of that specific collection in a table, allowing content investigation and properties of that collection.
(3) **Graph Filter** To filter for specific documents, users can apply conditions to node properties (Figure 2, F). The filtering supports various property data types such as boolean, number, string and date. When assigning multiple filter conditions for the same node object, all conditions must be met for a document to be returned, creating the effect of an "AND" operator. In order to create the effect of an "OR" operator instead, the vertex has to be duplicated on the canvas. An example for this is described in Section 4.

After finalizing the visual database query, users can initiate the graph retrieval (Figure 2, G). Subsequently, the visual database query is mapped to a textual query (more details in Section 3.4), which is then executed to retrieve a sub-graph from the database.

3.2 Ontology Graph

The ontology graph visually represents the underlying graph schema, a metagraph showcasing all entity types and their interconnections. This component facilitates a comprehensive understanding of the data model, empowering users to explore the connectivity of nodes through interactive actions like dragging, rotating, and zooming. This functionality is pivotal in helping users discern the types of queries they can model.

3.3 Graph Result Explorer

The graph result explorer visualizes the graph result of the user-defined visual query in three dimensions, offering interactive features such as dragging, zooming, and panning. It includes predefined view modes for different arrangements of nodes, enhancing the overall user experience. Users can explore properties of vertices and edges by clicking on and hovering over them, and the view enables the iterative expansion of neighbors for individual nodes. Also, it facilitates an option to export the whole graph or selected parts of it.

3.4 Implementation Design

Web-Interface

The front-end of *Graph Detective* is a web-application that runs on React v18[5]. The *Canvas Area for Query Modeling* is an interactive canvas built with the open-source library *React Flow*[6]. In React Flow, diagrams can be built using nodes and edges that are stored as JSON objects allowing for an efficient handling of the diagrams in both the front-end and back-end. As for the *Ontology Graph* and *Graph Result Explorer*, react-force-graph[7] is being used. It renders nodes and edges in an interactive 3D representation.

The back-end is built using the Python web application framework Flask[8]. It exposes various endpoints with which the front-end communicates via REST API calls. All functionality regarding the conversion of visual queries to ArangoDB AQL queries is implemented in the back-end.

The queries are generated iteratively by combining predefined sub-query templates. Typically, a final query first retrieves a set of *source documents*. Each one then initiates a separate graph traversal process to find full source-to-sink paths. We outline this process in more detail using the example visual query in Figure 5, which corresponds to the generated AQL query in Listing 1:

(1) Collect all *source documents* from the database. *Source documents* are those documents which - based on the visual query - do not have any incoming edges. In the example, these consist of all documents of type "Person" (lines 1-4).
(2) Compose an AQL graph traversal statement (lines 6 to 31). This statement contains a single FILTER statement (line 13) with multiple conditions. Any set of conditions separated by an "OR" operator (lines 14 to 22 "OR" lines 22 to 31) filters for a single, complete *source-to-sink* path (here: [`Person` → `Institution`] and [`Person` → `Project`]). Since graph traversals can only ever start traversing from a single document, we use a loop to repeat the traversal for each of the source documents (line 5).
(3) Return all paths (line 32). These are the ones which are rendered in the Graph Result Explorer (Section 3.3) eventually. Note that multiple distinct paths might share common documents. To display a single, coherent graph, we merge any document nodes that appear in multiple paths.

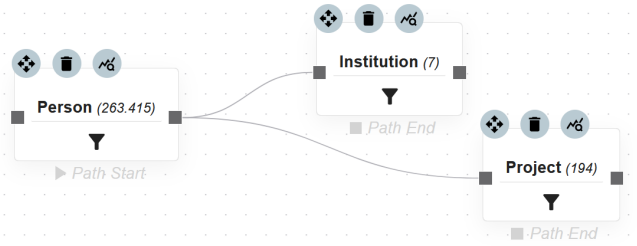


Figure 5: Example visual query from the user interface. The corresponding AQL query can be found in Listing 1.

```
1  LET source_nodes = FLATTEN(
2      FOR p IN Person
3          RETURN p
4  )
5  FOR source_node IN source_nodes
6      FOR v, e, p IN 0..1 ANY source_node
7      OPTIONS {
8          vertexCollections:
9              ["Person", "Institution"],
10         edgeCollections:
11             ["BelongsTo"]
12     }
13     FILTER (
14         (
15             (IS_SAME_COLLECTION(
16                 p.vertices[0], Person)
17             )
18             AND
19             (IS_SAME_COLLECTION(
20                 p.vertices[1], Institution)
21             )
22         ) OR (
23             (IS_SAME_COLLECTION(
24                 p.vertices[0], Person)
25             )
26             AND
27             (IS_SAME_COLLECTION(
28                 p.vertices[1], Project)
29             )
30         )
31     )
32     RETURN p
```

Listing 1: Generated AQL Query corresponding to Figure 5.

[5]https://react.dev/blog/2022/03/29/react-v18
[6]https://reactflow.dev/
[7]https://github.com/vasturiano/react-force-graph
[8]https://flask.palletsprojects.com/en/3.0.x/

4 USE CASE DEMONSTRATIONS

Any typical use case for property graphs (e.g., 360° view on enterprise data, which is often available in the form of huge document collections) benefits from the intuitive visual form of querying and displaying results that *Graph Detective* offers. As opposed to relational databases, graph databases are ideal for queries involving multi-hop relationships. If such queries were to be formulated on relational schema, the necessary use of table joins can quickly build up to a both complex and complicated query[9] *Graph Detective* can be used for both simple ad-hoc queries, as well as more sophisticated, multi-hop queries.

Our use cases deal with the analysis of discourses represented in document collections around the topic of *Climate Change* in science and society. We first collect documents covering several domains from four different data sources: Scientific Abstracts from **OpenAlex** [20] (~ 367K articles), **Wikipedia Page revisions** under the *Climate Change* category[10] (~ 1.2M revisions), **Greenpeace** articles curated by [3] (~ 700 articles), and **Climate Analytics** articles curated by [26] (~ 500 articles). These different sources cover broad domains including, respectively, science, community-driven content, specialized web and news articles [11].

The combined and processed graph consists of the texts of these documents, their metadata, and - most importantly - further language-related annotations like named entities, concepts, and emotions, as these can be used to reveal hidden semantic connections between documents. For creating this graph and annotations, we used the framework *Corpus Annotation Graph (CAG)* [10].

Keyword co-occurrence

The identification of common keywords across different articles mentioning a phrase like "wind energy" is of great interest. Modeling the visual query leads to finding the paths from respective `Text` nodes to their `Key Terms` (Figure 6, Query). After querying, the visual presentation shows commonly used key terms standing out (Figure 6, Graph). The most prominent terms and phrases are *solar energy*, *renewable energy* and *electricity*, while less used ones are *tidal power*, *indoor air*, and *depletion*. These insights allow to draw various conclusions, such as identifying which articles discuss the integration of which renewable energy source. In general, identifying such keywords allows to estimate prevalent topics, trends, and terminology used in climate change literature. Argubly, this use case could been handled by a relational database. Yet, *Graph Detective* is a practical choice for simple ad-hoc queries like this one.

Emotional Tone in News Articles

Next, we investigate the emotional tone conveyed by popular news sources around the topics of *environment* and *pollution*. It is sensible

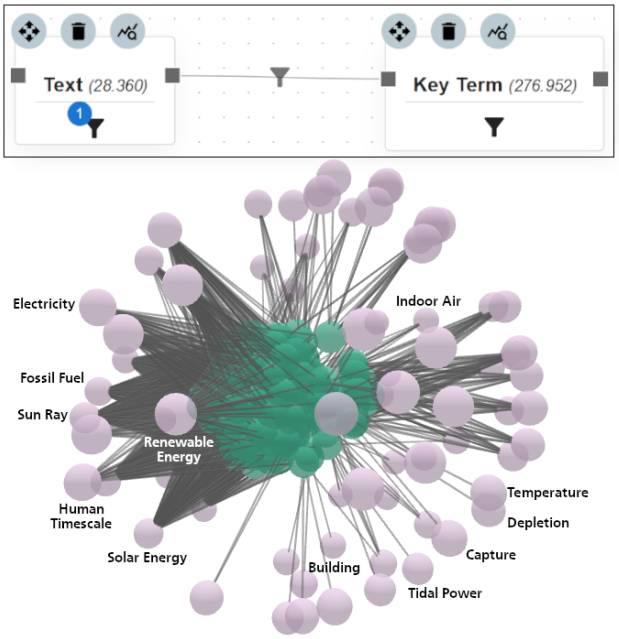


Figure 6: Query result for identifying common key terms in articles related to wind energy. Green: articles; purple: key term.

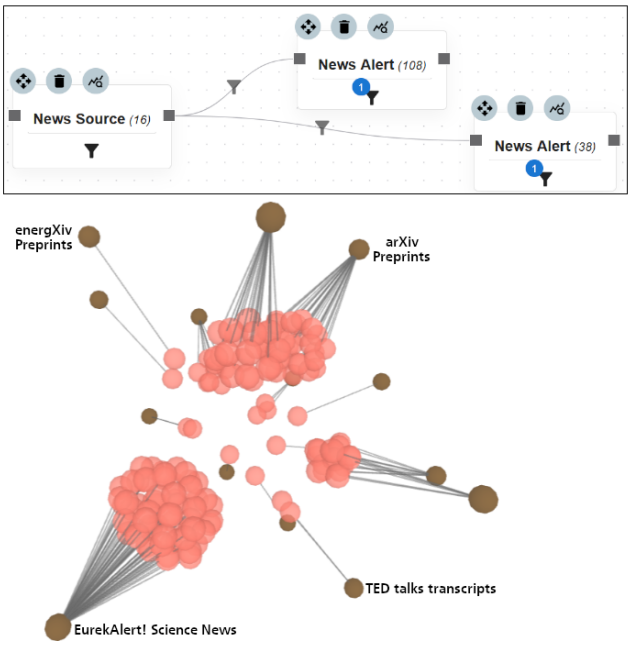


Figure 7: Query result for News Alerts by EurekAlert! Science News. Brown: News Source; Red: News Alert.

to break this process into separate subsequent queries. First, we retrieve the articles (*News Alerts*) that have been published by different *News Sources*. We filter only those articles whose title contains

[9] https://aws.amazon.com/compare/the-difference-between-graph-and-relational-database/?nc1=h_ls.
[10] https://en.wikipedia.org
[11] News articles were scraped from the following 14 sites: https://www.eurekalert.org/, https://arxiv.org/, https://www.nature.com/nature.rss, https://elib.dlr.de/, https://cdn.technologyreview.com/stories.rss, http://feeds.feedburner.com/carbonbrief, https://ec.europa.eu/commission/presscorner, https://pr.euractiv.com/pr/rss/all, https://techcrunch.com/feed/, https://www.ted.com/talks, https://engrxiv.org, https://www.die-gdi.de/rss2.xml, https://www.economist.com/science-and-technology/rss.xml, https://pubmed.ncbi.nlm.nih.gov/trending

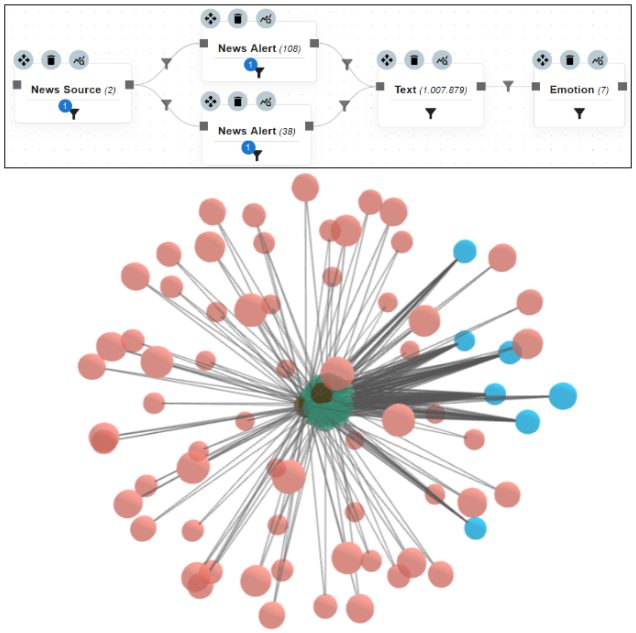


Figure 8: Emotions for articles from EurekAlert! Science News. Brown: News Source; Red: News Article; Green: Article Text; Blue: Emotion.

one or both of the terms *pollution* or *environment*. This scenario requires the modeling of an "OR" operator which is constructed by placing two distinct `News Alert` nodes on the canvas - one that filters for the term *pollution* and the other for *environment* (Figure 7, Query). After querying, solely from a visual perspective it already becomes obvious which News Source (here: `EurekAlert! Science News`) has the largest impact because of producing the largest number of news alerts (Figure 7, Graph).

Second, we filter articles published by `EurekAlert! Science News` and additionally display their corresponding `Emotion` annotation (Figure 8). After inspecting the emotions visually, it appears that 'neutral' and 'disgust' are prominent, whereas 'anger' and 'sadness' seem less present in those articles.

Based on our graph model, this query requires four distinct node types in a three-hop relationship, a typical scenario for graph databases.

5 EVALUATION

We conducted a user study to evaluate the effectiveness of GraphDetective. The main objective was to assess whether users unfamiliar with graphs and graph databases are able to access the example database using the interface.

5.1 Methodology

Participants

We selected eleven participants working in the field of Computer Science. Users responded with the following distribution on a 5-point Likert scale to questions addressing their previous knowledge:

"How frequently do you work with graph data?":
2x Never, 5x Rarely, 1x Occasionally, and 3x Frequently.

"How frequent do you design queries for graph databases?":
8x Never, 3x Rarely.

Study Design and Procedure

After assessing their baseline knowledge, participants received a standardized description of the web interface and the underlying graph database to ensure an equal understanding. Content-related questions were not answered during the study and participants were not given any feedback on their results before they finished completing all three tasks. All participants were previously unfamiliar with the interface as well as the database that it is connected to.

Each participant completed three sequential tasks using the web interface. Each task asks the participant to find specific information, such as identifying articles on a particular topic or listing authors within a specific field or affiliation. Tasks increased in difficulty based on factors such as the number of nodes, edges and filters required to complete the task successfully. We split participants into two groups with a different set of questions to increase diversity.

After each task, participants provided feedback via questionnaires covering interface intuitiveness, solution confidence, the use and intention of certain functionalities and any encountered difficulties. Task completion time was also recorded. Correctness of solutions was evaluated based on whether the query result contained the requested information and focuses on only the relevant data elements without excessive output.

5.2 Results

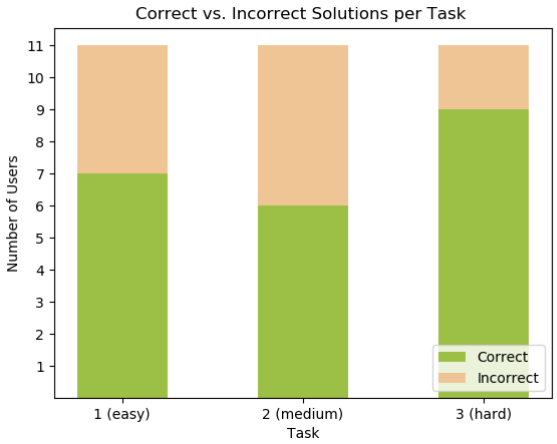
Task fulfilment

The core benefit of GraphDetective is that users can avoid writing textual database queries (here: Arango AQL queries) manually and instead use visual means. Indeed, as shown in Figure 9a, most participants were able to successfully solve the tasks in this way, which would not have been the case without GraphDetective. Notably, the success rate for the last, most complex task, was clearly higher than for the first two tasks. A likely reason is that the participants gained familiarity with the interface while progressing through the earlier, easier tasks. This is corroborated by the observation, that the median time spent on the first task was 11 minutes compared to 6 minutes for the last task (Figure 9b).

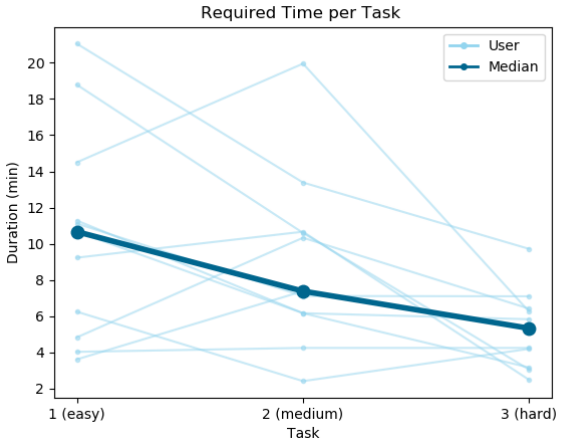
Qualitative feedback

While the user study showed the general usefulness of GraphDetective, the detailed qualitative feedback of the users provided more information of how users apply the tool and what they are missing. Throughout the tasks, the participants indicated a subtle increase in intuitiveness of the interface but also a slight decrease in confidence with their results.

All participants made frequent use of both the Graph Ontology (Figure 3) and Document Viewer (Figure 2, `E`) and stated that it helped them complete the task in most of the cases. The Ontology



(a) Number of correct vs. incorrect solutions for each task.



(b) Time taken (minutes) to complete each task.

Figure 9: Evaluation Results

was mainly used for understanding the relationships between entities. The Document View was mainly used to understand the data format of specific nodes and identify specific attributes and values for subsequent filtering.

There appeared to be a frequent misconception about the Document Viewer (Figure 2, E) function. Many participants were looking for intermediate query results in the table of the Document Viewer and some were expecting the final result to show up in this table.

Furthermore, the users expressed some confusion about the correct use of the ontology graph. Also, a majority of users stated that the distinction between AND and OR operators for multiple node filters was unclear to them.

Notably, we encountered unexpected circumstances where users attempted to apply relational database concepts to the logic of the graph database. Some of the responses revealed that users were sometimes looking for common node properties in connected nodes in an attempt to create a foreign key, as it is typically the case for relational databases. This seems to be a fundamental misunderstanding by some participants, as the underlying logic is based on graphs rather than relational databases. Yet, it is an additional explanation for lacking confidence and confusions about the proper usage of the Ontology (Figure 3).

Implications for the future interface design

By far the biggest confusion originated from the Document Viewer (Figure 2, E) function. This tabular view was originally designed for browsing through documents of individual nodes. However, the majority of users assumed that this view would either display final query results, intermediate query results, or refresh automatically after applying a conditional filter to a node.

While it is not intended to show final query results in this particular view, we realize that the view can be made more interactive. We think that a feasible solution includes options to add and delete filters through this tabular view and automatically update its content to match the selected node filters. Ideally, the two tabs "Node Analyzer" and "Filters" will be combined into a single tab menu, which further emphasizes their functional similarities. Showing intermediate query results updating live while the user designs the visual query would be very useful for the user but is challenging to implement, as it would involve constant execution of the query. Especially for large graphs, this will cause extended delays and computational overhead.

Furthermore, the current method of combining multiple filters with either an "OR" or an "AND" operator lacks intuitiveness. In the present version, simply adding multiple filters to the same node results in an "AND" operator, meaning that all conditions must be met for a document to be returned. To create the effect of an "OR" operator, the node must be duplicated on the canvas instead. In a future version, this functionality should be implemented in a single view, reducing not only confusion but also unnecessary node objects that potentially clutter the canvas area.

Finally, it can be argued that the pool of evaluators might not be representative of the potential users who would most benefit from this tool. Our user study comprised of evaluators working in the field of Computer Science yet unfamiliar with graph database concepts. *Graph Detective* has the potential of being useful also to users without such technical background, but we found that its intuitiveness should be further increased, for example by providing an interactive walk-through with example queries.

6 FUTURE WORK

Several lines of future work could add major benefits to *Graph Detective*.

Automated Query Support

Currently, the interface puts the user in charge of building the complete query. They fully decide the nodes, properties and edges to model on the canvas without significant external support. Therefore, automated exploitation of the graph schema during query design can decrease modeling time and increase the accuracy of the user's query. This can be accomplished by adding features such as node and edge auto-completion.

Graph Projections

When a user executes a query, it returns the raw resulting graph. However, there are cases where additional nodes need to be queried to retrieve the desired output. In future work, we plan to address this by allowing users to hide specific nodes in the result. For example, in Figure 8, a user might only want to see the `Emotions` linked to articles by specific `News Sources`. Due to the graph schema, it is currently necessary to query intermediate nodes like `News Alert` and `Text`, but there is no option to hide these nodes in the visualization.

Language Model Guided Query Design

With advancements in generative Artificial Intelligence, query modeling can be performed by Tool-Augmented Language Models [19, 21, 30]; Language Models that have the ability to manipulate the users query based on their natural language input. This would greatly reduce the extent to which the user is required to understand the database model.

7 CONCLUSION

The adoption of graph databases across diverse domains has highlighted their efficiency in storing graph data. However, querying such graphs, particularly for those not well familiar with database concepts, remains a significant challenge.

To address this issue, we presented *Graph Detective*, an interface that provides visual means to query ArangoDB databases and allows a 3D interaction with the graph database. Our interface empowers users to create visual database queries, effectively querying ArangoDB without needing expertise in database query writing. Also, our interface allows users to export graph results and thus link them to external tools to enhance analytical capabilities. We verified the usefulness of our concept and application through a user study and revealed that even individuals inexperienced with graph databases could successfully solve the tasks of the study.

8 ACKNOWLEDGEMENTS

This project is funded by the German Federal Ministry of Education (BMBF) under the InsightsNet project, grant no. 01UG2130A.

REFERENCES

[1] Renzo Angles and Claudio Gutierrez. 2008. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40, 1, 1–39.

[2] Artur Baranowski and Nico Hochgeschwender. 2021. Grammar-constrained neural semantic parsing with LR parsers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors. Association for Computational Linguistics, Online, (Aug. 2021), 1275–1279. DOI: 10.18653/v1/2021.findings-acl.108.

[3] Sabine Bartsch, Changxu Duan, Sherry Tan, Elena Volkanovska, and Wolfgang Stille. 2023. The insightsnet climate change corpus (iccc).

[4] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. 2009. Gephi: an open source software for exploring and manipulating networks. (2009). http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154.

[5] Aleksandar Bobic, Jean-Marie Le Goff, and Christian Gütl. 2023. Exploring tabular data through networks. In *Advances in Information Retrieval*. Jaap Kamps, Lorraine Goeuriot, Fabio Crestani, Maria Maistro, Hideo Joho, Brian Davis, Cathal Gurrin, Udo Kruschwitz, and Annalina Caputo, editors. Springer Nature Switzerland, Cham, 195–200. ISBN: 978-3-031-28241-6.

[6] 2018. *Introduction. Querying Graphs*. Springer International Publishing, Cham, 1–2. ISBN: 978-3-031-01864-0. DOI: 10.1007/978-3-031-01864-0_1.

[7] Hejie Cui et al. 2023. A survey on knowledge graphs for healthcare: resources, applications, and promises. (2023). arXiv: 2306.04802 [cs.AI].

[8] Jose Danado and Fabio Paternò. 2014. Puzzle: a mobile application development environment using a jigsaw metaphor. *J. Vis. Lang. Comput.*, 25, 4, (Aug. 2014), 297–315. DOI: 10.1016/j.jvlc.2014.03.005.

[9] Alin Deutsch, Yu Xu, Mingxi Wu, and Victor Lee. 2019. Tigergraph: a native mpp graph database. *arXiv preprint arXiv:1901.08248*.

[10] Roxanne El Baff, Tobias Hecking, Andreas Hamm, Jasper W. Korte, and Sabine Bartsch. 2023. Corpus annotation graph builder (CAG): an architectural framework to create and annotate a multi-source graph. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. Danilo Croce and Luca Soldaini, editors. Association for Computational Linguistics, Dubrovnik, Croatia, (May 2023), 248–255. DOI: 10.18653/v1/2023.eacl-demo.28.

[11] Yanlin Feng, Xinyue Chen, Bill Yuchen Lin, Peifeng Wang, Jun Yan, and Xiang Ren. 2020. Scalable multi-hop relational reasoning for knowledge-aware question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors. Association for Computational Linguistics, Online, (Nov. 2020), 1295–1309. DOI: 10.18653/v1/2020.emnlp-main.99.

[12] Rita Francese, Michele Risi, and Genoveffa Tortora. 2016. Iconic languages: towards end-user programming of mobile applications. *Journal of Visual Languages & Computing*, (Nov. 2016). DOI: 10.1016/j.jvlc.2016.10.009.

[13] Tom Freeman, Sebastian Horsewell, Anirudh Patir, Josh Harling-Lee, Tim Regan, Barbara Shih, James Prendergast, David Hume, and Tim Angus. 2022. Graphia: a platform for the graph-based visualisation and analysis of high dimensional data. *PLOS Computational Biology*, 18, (July 2022), e1010310. DOI: 10.1371/journal.pcbi.1010310.

[14] Manas Gaur, Keyur Faldu, and Amit Sheth. 2021. Semantics of the black-box: can knowledge graphs help make deep learning systems more interpretable and explainable? *IEEE Internet Computing*, 25, 1, 51–59. DOI: 10.1109/MIC.2020.3031769.

[15] Björn A. Johnsson and Boris Magnusson. 2020. Towards end-user development of graphical user interfaces for internet of things. *Future Gener. Comput. Syst.*, 107, C, (June 2020), 670–680. DOI: 10.1016/j.future.2017.09.068.

[16] Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. 2021. Characterizing visual programming approaches for end-user developers: a systematic review. *IEEE Access*, 9, 14181–14202. DOI: 10.1109/ACCESS.2021.3051043.

[17] Yiheng Liu et al. 2023. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, 1, 2, (Sept. 2023), 100017. DOI: 10.1016/j.metrad.2023.100017.

[18] Dominik Opitz and Nico Hochgeschwender. 2022. From zero to hero: generating training data for question-to-cypher models. In *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*, 17–20. DOI: 10.1145/3528588.3528655.

[19] Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: tool augmented language models. (2022). DOI: 10.48550/arXiv.2205.12255.

[20] Jason Priem, Heather Piwowar, and Richard Orr. 2022. Openalex: a fully-open index of scholarly works, authors, venues, institutions, and concepts. (2022). arXiv: 2205.01833 [cs.DL].

[21] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. (2023). DOI: 10.48550/arXiv.2302.04761.

[22] Phillip Schneider, Tim Schopf, Juraj Vladika, Mikhail Galkin, Elena Simperl, and Florian Matthes. 2022. A decade of knowledge graphs in natural language processing: a survey. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Yulan He, Heng Ji, Sujian Li, Yang Liu, and Chua-Hui Chang, editors. Association for Computational Linguistics, Online only, (Nov. 2022), 601–614. https://aclanthology.org/2022.aacl-main.46.

[23] [n. d.] Scratch - imagine, program, share. https://scratch.mit.edu/. Accessed: 2010-11-18. ().

[24] Salvatore Sorce, Alessio Malizia, Vito Gentile, Pingfei Jiang, Mark A. Atherton, and David Harrison. 2019. Evaluation of a visual tool for early patent infringement detection during design. In *International Symposium on End-User Development*. https://api.semanticscholar.org/CorpusID:195785537.

[25] Srikanth G Tamilselvam, Naveen Panwar, Shreya Khare, Rahul Aralikatte, Anush Sankaran, and Senthil Mani. 2019. A visual programming paradigm for abstract deep learning model development. In *Proceedings of the 10th Indian Conference on Human-Computer Interaction* (IndiaHCI '19) Article 16. Association for Computing Machinery, Hyderabad, India, 11 pages. ISBN: 9781450377164. DOI: 10.1145/3364183.3364202.

[26] Elena Volkanovska, Sherry Tan, Changxu Duan, Paul Chowdhury Debajyoti, and Sabine Bartsch. 2023. Presenting an annotation pipeline for fine-grained linguistic analyses of multimodal corpora. In *Proceedings of the 19th Conference on Natural Language Processing (KONVENS 2023)*. KONVENS 2023 Organizers, Ingolstadt, Germany, (Dec. 2023).

[27] Chengbin Wang, Xiaogang Ma, Jianguo Chen, and Jingwen Chen. 2018. Information extraction and knowledge graph construction from geoscience literature. *Computers & geosciences*, 112, 112–120.

[28] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, and Vadim Sheinin. 2018. Graph2seq: graph to sequence learning with attention-based neural networks. *CoRR*, abs/1804.00823. http://arxiv.org/abs/1804.00823 arXiv: 1804.00823.

[29] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. 2023. Harnessing the power of llms in practice: a survey on chatgpt and beyond. (2023). arXiv: 2304.13712 [cs.CL].

[30] Jiawei Zhang. 2023. Graph-toolformer: to empower llms with graph reasoning ability via prompt augmented by chatgpt. (2023). DOI: 10.48550/arXiv.2304.11116.

[31] Ningyu Zhang, Shumin Deng, Zhanlin Sun, Guanying Wang, Xi Chen, Wei Zhang, and Huajun Chen. 2019. Long-tail relation extraction via knowledge graph embeddings and graph convolution networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Jill Burstein, Christy Doran, and Thamar Solorio, editors. Association for Computational Linguistics, Minneapolis, Minnesota, (June 2019), 3016–3025. DOI: 10.18653/v1/N19-1306.