



KGNav: A Knowledge Graph Navigational Visual Query System

Xiang Wang
Tianjin University
Tianjin, China
xiang_w@tju.edu.cn

Zhaozhuo Li
Tianjin University
Tianjin, China
lizhaozhuo@tju.edu.cn

Xin Wang
Tianjin University
Tianjin, China
wangx@tju.edu.cn

Dong Han
Tianjin Academy of Fine Arts
Tianjin, China
winter1976@hotmail.com

ABSTRACT

Visual query is a vital technique for comprehending and analyzing knowledge graphs, which provides an effective method to lower the barrier of querying knowledge graphs for non-professional users. Nevertheless, visual query techniques for knowledge graphs and ontologies that have emerged in recent years cannot bridge the gap between global information provided by the knowledge graph schema and underlying data of knowledge graph. Thus it cannot fully exploit the global information to navigate users for querying knowledge graphs. This demonstration showcases KGNav, a Knowledge Graph Navigational visual query system. KGNav (1) redefines the minimal unit of operation to abstract the conceptual hierarchy, i.e., Knowledge Graph Schema, in the domain from the original knowledge graph in an offline semi-automatic way through the equivalence relations between these units; it also (2) provides a series of operators and an interactive GUI to capture user query intentions, guiding users to explore the Knowledge Graph Schema to achieve in-depth analysis of knowledge graphs. We will demonstrate the capability of KGNav in reducing tedious queries, enabling users to swiftly grasp the structure of the knowledge graph, and performing queries through several fundamental scenarios.

PVLDB Reference Format:

Xiang Wang, Xin Wang, Zhaozhuo Li, and Dong Han. KGNav: A Knowledge Graph Navigational Visual Query System. PVLDB, 16(12): 3946 - 3949, 2023.
doi:10.14778/3611540.3611592

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/XiangLovin/KGNav>.

1 INTRODUCTION

Querying is a vital tool for data exploration in the era of big data. With Knowledge Graphs (KG) having been widely adopted in various domains, an effective and efficient querying mechanism on large-scale KGs, e.g., DBpedia and Wikidata, is an essential requirement for users' evolving exploration demand over KGs to find the

information of their interests. Hence, how to effectively, quickly, and accurately query such KGs with diverse and complex patterns poses a significant challenge to the KG community. And thus far, the most representative query languages for KGs are SPARQL and Cypher, for RDF graphs and property graphs, respectively. However, using such KG query languages is quite challenging for non-professional users. On one hand, users have to learn sophisticated techniques to customize expressive queries using these languages; on the other hand, it is difficult for users to construct useful query patterns without the guidance from KG structure information.

In recent years, visual query methods of KGs for non-professional users have emerged. RDF-GL [2] and QueryVOWL [1] are visual query languages that map syntax symbols to graphical elements and thus enhance user-friendliness to some extent, however, which cannot support logical combinations of filtering criteria. The keyword-based query system QQBE [3] simplifies the user input and the need of background knowledge about KG, but can only perform simple tuple pattern queries, which cannot precisely express users' query intention. The facet-based query system Grafa [5] can make the final results meet the user query intention, but can only support simple star-shaped query patterns. VISAGE [7] constructs query graph templates with different types of nodes representing different ontology instances, which can partially reflect the associations between entities and simplify the construction process of query patterns, nevertheless, the limited number of templates hinders fine-grained expression of users' query intention.

However, the existing visual query methods merely focus more on the detailed construction process of query patterns, which ignores global information in schema layers, i.e., ontology layer, of KGs. Meanwhile, it is hard for users to understand overview of KGs due to rapid update of underlying data. Although ontology visualization tools, such as Protégé [6] and OntoPlot [8], can help to show hierarchical relations of ontologies, they only visualize schema of KGs and neglect the data layers, causing the difficulty for users to leverage ontologies to guide their queries on KGs. Therefore, in the state-of-the-art KG visual query methods, there is an obvious gap between the schema layer and the data layer.

To this end, we propose KGNav, a novel navigational interactive query system, which can guide users to easily understand structures of KGs and conduct queries on KGs by constructing a visualized Knowledge Graph Schema (KGS). The core idea of KGNav is to first redefine the minimum basic operation unit of KGS, then use

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611592

the equivalence relation between these units to automatically extract the conceptual hierarchy in a domain, and thus condense the original KG to a much smaller scale that can be perceived by users at a glance. We call this condensed graph “*schema graph*”, which achieves a degree of domain knowledge reuse. In another words, when users encounter a new KG, the constructed schema graph, i.e., the instance of a KGS, will guide them to easily recognize the structure of KG, maximize the efficiency of knowledge sharing, and reduce the redundancy of knowledge.

In addition, we offer a succinct and user-friendly interactive graphical user interface (GUI) and a series of useful operators, which enable users to comprehend schema of KGs and execute query operations on underlying KG data, thus effectively utilizing the hierarchical relations in schema layer (refer to Sec 2.2) while keeping the query processing capability in data layer. Our demonstration visualizes various query intentions that users may request, provides three scenarios, and showcases the well performance of KGNV in guiding users to successfully construct queries of their interests. The system can be accessed and manipulated through the website (<http://152.136.45.252/kgnav>).

2 KGNV OVERVIEW

Figure 1 shows the system architecture of KGNV, which consists of three main components, i.e., AGG, KGSC, and SN, and an interactive graphical user interface (GUI). We will introduce the core ideas of each component and their technical contributions.

2.1 Atomic Graph Generator (AGG)

AGG plays a vital role in cleansing and structuring the data of KGs. In the KGS, each entity node is termed as “*hypernode*”, which is essentially an entity recursively consisting of a set of other hypernodes with equivalence relations. When AGG first obtains raw data of a KG, each entity in the KG, by itself, is identified as a hypernode, so that the KG can be wrapped as a KGS, ensuring that AGG performs consistent operations when processing KG or KGS data.

We define “*atomic graph*” as a primitive operation unit in KGNV. An atomic graph consists of a hypernode and its entire set S of incoming and outgoing edge labels. In a KGS, each hypernode uniquely maps to an atomic graph and the labels in S constitute the feature of the hypernode. For instance, given an entity e , i.e., a hypernode, its n incoming edges form the incoming edge label set $L_{in} = \{a_1, a_2, \dots, a_n\}$ and its m outgoing edges form the outgoing edge set $L_{out} = \{b_1, b_2, \dots, b_m\}$, respectively, then the semantic description of its atomic graph can be defined as $A(e) = \langle e, F \rangle$, where $F = \langle L_{in}, L_{out} \rangle$ jointly form the feature of $A(e)$.

After each round of inputting a KG (or KGS) G into AGG, a structured data will be outputted, which primarily consists of a set of atomic graphs, and their corresponding edges in G , respectively.

2.2 KGS Constructor (KGSC)

The schema is an important data modelling facility to define meta-data. In relational databases, a schema defines all structural aspects of a database, including relation and attribute names, attribute types, and various other constraints on relations. For KGs, a schema is an explicit specification of the structural constraints on entities (nodes) and relationships (edges) in a KG. Generally, a KG schema is quite

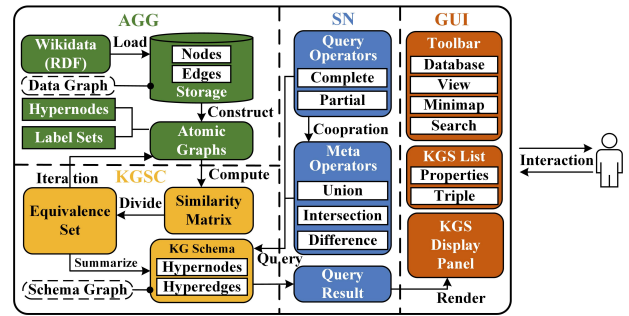


Figure 1: The overall architecture of KGNV

helpful for users to have an overall understanding of a KG (esp. of large-scale). However, unlike a schema of a relational database, a schema of a KG is not necessarily a requirement. While, it is labor-intensive and error-prone for non-expert users to manually construct a schema over an existing large-scale KG. Therefore, KGSC adopts an iterative “bottom-up” construction method to help users construct a schema, i.e., KGSC is iteratively utilized to abstract and aggregate atomic graphs generated by AGG, so that the scale of final generated KGS will be much smaller than that of original KG (see AGG and KGSC in Figure 1).

More specifically, given a KG G , after data processing of AGG, a set of atomic graphs A_1 will be forwarded to KGSC. Then, KGSC automatically computes and generates a similarity matrix M_1 , which is used to classify the hypernodes in A_1 into different sets of equivalent hypernodes, based on the equivalence similarity between any two hypernodes in A_1 . Each set of equivalent hypernodes is then abstracted into a new hypernode to generate a new KGS D_1 . Then, KGSC passes D_1 back to AGG to generate a new set of atomic graphs A_2 , and continue for the next iteration. Except for A_1 , which is derived from KG G , A_i ($i = 2, \dots, k$) are derived from KGS D_{i-1} generated by KGSC in the previous iteration. The whole process ends when A_i no longer changes for the consecutive rounds of iterations, and we refer to the final generated KGS D_i as the “*schema graph*” \mathcal{G} , which is abstracted from the original “*data graph*” G .

Subsequently, the final schema graph \mathcal{G} will be forwarded to the GUI of KGNV for rendering. It allows users to more intuitively understand and manipulate the generated KGS \mathcal{G} , which is a more succinct graph compared to the original KG G .

2.3 Schema Navigator (SN)

The Schema Navigator (SN) is a crucial component of KGNV that is directly related to the users’ interaction with KGNV, whose theoretical foundation is derived from KGVQL [4]. By defining operators that can be manipulated by users, KGNV can implement the set of queries on KGs introduced in KGVQL, such that different query intentions of users can be satisfied. These operators include:

Meta operators: Meta operators consist of “*union*”, “*intersection*”, and “*difference*”, which are applied on atomic graphs. Their primary functions are to execute simple queries on hypernodes, and the results that can be reused in the subsequent queries. Figure 2(a) shows the graphical and semantic descriptions of meta operators, denoted by **OP**. Let $A(N_1)$ and $A(N_2)$ denote the atomic graphs


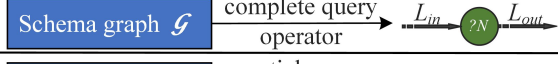
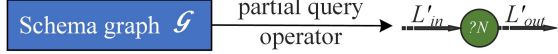
Type	Graphical Description	Semantic Description
(a) Meta operators		$A(N_1) \text{ OP } A(N_2) = A(N_1 \text{ OP } N_2) = A(N)$ $= \langle N, \langle L_{in}, L_{out} \rangle \rangle$
(b) Query operators		$\langle ?N, \langle L_{in}, L_{out} \rangle \rangle$
		$\langle ?N, \langle L'_{in}, L'_{out} \rangle \rangle$ ($L'_{in} \in \mathcal{L}^*_{in}, L'_{out} \in \mathcal{L}^*_{out}$)

Figure 2: Graphical and semantic descriptions of meta operators and query operators in Schema Navigator

of hypernodes N_1 and N_2 , respectively, L_{in_i} and L_{out_i} ($i = 1, 2$) indicate the incoming and outgoing edge label set of each hypernode, respectively. A newly generated hypernode $N = N_1 \text{ OP } N_2$ is a set of hypernodes constructed by operating on the original KG that N_1 and N_2 correspond to, respectively, using meta operators. And the feature $\langle L_{in}, L_{out} \rangle$ of its atomic graph $A(N) = \langle N, F \rangle$ is extracted from the entire sets of incoming and outgoing edge labels of hypernodes in N .

Query operators: Query operators are applied on a schema graph \mathcal{G} , and include of “complete query operator (cqo)” and “partial query operator (pqo)”, which are depicted in Figure 2(b). Let $A(?N)$ denote the atomic graphs of hypernodes in results variable $?N$, and suppose that the expected feature of $A(?N)$ is $F = \langle L_{in}, L_{out} \rangle$ and that \mathcal{L}^*_{in} and \mathcal{L}^*_{out} are Kleene closures of L_{in} and L_{out} , respectively. For the cqo, $?N$ contains all hypernodes in the data layer whose feature of $A(?N)$ is $F = \langle L_{in}, L_{out} \rangle$, while the pqo only requires that features of atomic graphs corresponding to hypernodes in $?N$ match $\langle L'_{in}, L'_{out} \rangle$, where $L'_{in} \in \mathcal{L}^*_{in}$ and $L'_{out} \in \mathcal{L}^*_{out}$.

Meta operators allow users to quickly construct simple set-based queries, which provides more possibilities for reusing query results. In addition, query operators adopt faceted search to allow users to query with edge labels, which further compensates for the shortcomings of meta operators that are unable to effectively utilize edge information. By leveraging both types of operators, KGNav can dynamically record user actions and automatically process the underlying KG to give users feedback. These two types of operators reduce the difficulties in constructing complex queries through simple operations, which allows users to flexibly adjust query orders and combine query results according to their specific needs and query intentions, which is not available in the existing KG (or ontology) visual methods.

3 DEMONSTRATION SCENARIOS

KGNav employs Vue framework and Ant Design, AntV G6 graphical visualization components to implement a friendly GUI and all the aforementioned interactive effects. For our demonstration, we use the interactive GUI of the snapshot, as shown in Figure 3, which simulates three typical query scenarios to reflect diverse query intentions of users in KG queries. We guide the participants to operate and experience KGNav according to the following steps.

KGS Exploration. To make full use of the structural information offered by KGS, we can locate the entity “Michael Jackson” by steps described with the green labels in Figure 3.

① Hover the mouse over the hypernode “human” to highlight the associated edges and hypernodes. We can find the hypernode “singer” under “human” in the Content Preview Panel.

② Choose one of the three methods to access “singer”: right-click on “human” to expand, click “singer” in the Content Preview Panel, or click and expand “human” on the KGS List Panel.

③ Keep exploring until finding and clicking on “Michael Jackson”. All the properties and triples of “Michael Jackson” will be displayed in the Entity Information Panel. Meanwhile, all the hypernodes that directly include “Michael Jackson” will be highlighted in the KGS Display Panel.

④ Click on a triple in the Entity Information Panel to view its corresponding hypernodes and edges in the KGS Display Panel.

Thus, these four steps constitute a round of KGS exploration. If users would like to specify a target entity, they are allowed to enter “Michael” or “MJ” in the search bar on the top of the KGS List Panel. KGNav will perform fuzzy search and provide options for users to directly locate “Michael Jackson”.

Simple Query. It is obvious that the steps in above KGS Exploration alone are insufficient to satisfy various users’ query intentions. Therefore, we can use “meta operators” to construct more expressive queries according to the following steps, which are denoted by the blue labels in Figure 3.

① The process usually starts with two hypernodes. After expanding the hypernode “human”, hold down the “Shift” key, and simultaneously select two hypernodes “singer” and “actor”. The Meta Operator Panel will become active.

② Click on the “union” operator on the Meta Operator Panel to retrieve all singers and actors in the KG. Alternatively, we can also select “intersection” operator to retrieve individuals (entities) who are both singers and actors.

③ KGNav will generate a new hypernode, which will be automatically labeled as “singer or actor”, on the KGS Display Panel.

④ Query results are reusable, e.g., using “singer or actor” and “director” to execute the “difference” operation can also retrieve all the actors or singers who are not directors.

These four steps constitute a round of simple query. The result of a query can be retained as a hypernode in the KGS and the steps in KG Exploration can be recursively performed on this hypernode.

Complex Query. Next, we show how to use “query operators”, by the following steps in the red labels, to query with edge labels and gradually expand from simple to complex queries.

① In the Query Operator Panel, select the designated node category “All” to start querying. The system will automatically generate suggested label based on the selection.

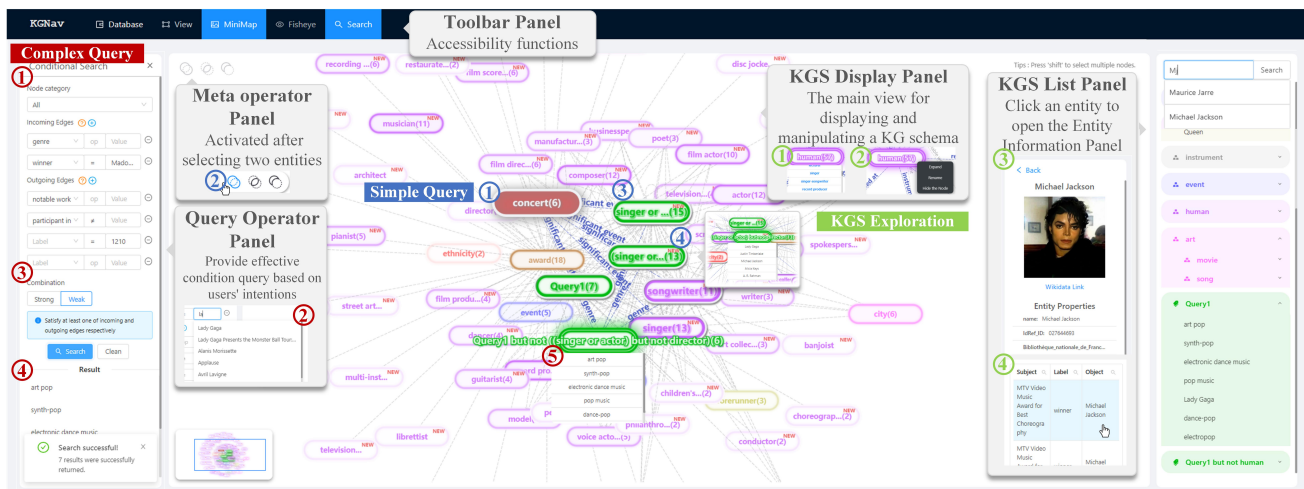


Figure 3: KGNav User Interface: main panels and demonstration scenarios

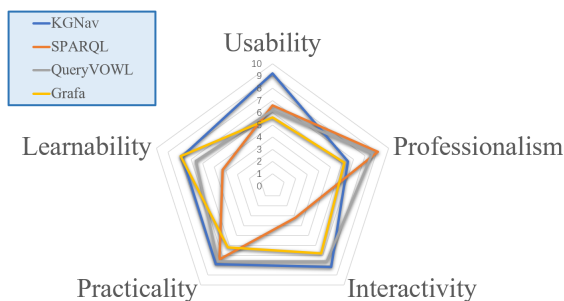


Figure 4: User Engagement Result

② Continue to specify any number of incoming and outgoing edge labels as an expected atomic graph feature, while KGNav can automatically filter out invalid conditions.

③ Select “Weak”, i.e. partial query operator, as the query operator to be used for querying. the system will automatically construct a target atomic graph pattern to get the result match from the schema graph in the KGS Display Panel.

④ The results will be presented in the Result Panel. Moreover, the KGS Display Panel and the KGS List Panel will synchronously display this result as a hypernode named “Query1”.

⑤ Perform an “intersection” operation on the results produced in the previous steps, i.e., “Query1” and “(singer or actor) but not director”, can obtain all the actors or singers who are not directors and satisfy the current condition query “Query1”.

These results can be arbitrarily combined according to users’ query intentions, allowing the generation of arbitrarily complex queries, and enabling to stop a query when it becomes meaningless.

Demonstration Engagement. We recruited non-professional participants to accomplish several predefined query tasks using KGNav and three other KG query methods (i.e., SPARQL, Grafa, and QueryVOWL) for system evaluation. Regarding these four methods, user experiences are highly dependent on the their own subjectivity, and since the existing KG schema evaluations lack standard

benchmarks, we resort to questionnaires to assess the following dimensions of the systems: usability, professionalism, interactivity, practicality, and learnability. The results are illustrated in Figure 4. It is evident that KGNav is inferior to SPARQL and QueryVOWL in more complex and professional query capabilities, but it is more user-friendly and easy to learn, and has higher practicality than ordinary visual query methods. Therefore, the system evaluation have verified the advantage of KGNav in facilitating users to understand KGs in a more comprehensive way.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (2019YFE0198600), and the National Natural Science Foundation of China (61972275). We also thank Huawei for contribution. Xin Wang is the corresponding author of this paper.

REFERENCES

- [1] Florian Haag, Steffen Lohmann, Stefan Siek, and Thomas Ertl. 2015. QueryVOWL: A Visual Query Notation for Linked Data. In *The Semantic Web. Latest Advances and New Domains*, Vol. 9341. 387–402.
- [2] Frederik Hogenboom, Viorel Milea, Flavius Frasinca, and Uzay Kaymak. 2010. RDF-GL: A SPARQL-Based Graphical Query Language for RDF. In *Emergent Web Intelligence: Advanced Information Retrieval*. 87–116.
- [3] Nandish Jayaram, Arijit Khan, Chengkai Li, Xifeng Yan, and Ramez Elmasri. 2015. Querying Knowledge Graphs by Example Entity Tuples. *IEEE Transactions on Knowledge and Data Engineering* 27, 10 (2015), 2797–2811.
- [4] Pengkai Liu, Xin Wang, Qiang Fu, Yajun Yang, Yuan Fang Li, and Qingpeng Zhang. 2022. KGVQL: a knowledge graph visual query language with bidirectional transformations. *Knowledge-Based Systems* 250 (2022), 108870.
- [5] José Moreno-Vega and Aidan Hogan. 2018. GraFa: Scalable Faceted Browsing for RDF Graphs. In *The Semantic Web – ISWC 2018*, Vol. 11136. 301–317.
- [6] N.F. Noy, R.W. Ferguson, and M.A. Musen. 2000. The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. In *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, Vol. 1937. 17–32.
- [7] Robert Pienta, Acar Tamersoy, Alex Endert, Shankant Navathe, Hanghang Tong, and Duen Horng Chau. 2016. VISAGE: Interactive Visual Graph Querying. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI’16)*. 272–279. <https://doi.org/10.1145/2909132.2909246>
- [8] Ying Yang, Michael Wybrow, Yuan-Fang Li, Tobias Czauderna, and Yongqun He. 2020. OntoPlot: A Novel Visualisation for Non-hierarchical Associations in Large Ontologies. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 1140–1150.