VERTIGo: a Visual Platform for Querying and Exploring Large Multilayer Networks

Erick Cuenca, Arnaud Sallaberry, Dino Ienco, and Pascal Poncelet

Abstract—Many real world data can be modeled by a graph with a set of nodes interconnected to each other by multiple relationships. Such a rich graph is called multilayer graph or network. Providing useful visualization tools to support the query process for such graphs is challenging. Although many approaches have addressed the visual query construction, few efforts have been done to provide a contextualized exploration of query results and suggestion strategies to refine the original query. This is due to several issues such as i) the size of the graphs ii) the large number of retrieved results and iii) the way they can be organized to facilitate their exploration. In this paper, we present *VERTIGo*, a novel visual platform to query, explore and support the analysis of large multilayer graphs. *VERTIGo* provides coordinated views to navigate and explore the large set of retrieved results at different granularity levels. In addition, the proposed system supports the refinement of the query by visual suggestions to guide the user through the exploration process. Two examples and a user study demonstrate how *VERTIGo* can be used to perform visual analysis (query, exploration, and suggestion) on real world multilayer networks.

Index Terms—Visual Querying System, Visual Pattern Suggestion, Multilayer Networks

1 INTRODUCTION

Graphs are ubiquitous structures that capture relationships (edges) among entities (nodes). Many real world data can be modeled by a graph where multiple types of entities are connected to each other by multiple types of relationships. A *multilayer network*, or *multilayer graph*, allows the definition of multiple types of nodes and edges [1]. Examples of multilayer graphs are: social networks spanning over the same set of people, but involving different life aspects (e. g., social relationships - Facebook, Twitter, Google+, work relationships - LinkedIn); protein-protein interaction networks where each layer represents a different pathway on which proteins interact [2]; bibliographic networks where nodes are authors and the edge type spans over the set of conferences/journals [3]; knowledge graphs (i. e. Resource Description Framework) where the same subject/object nodes pair is connected by different predicates [4].

Considering the significance of such structure to represent and model real world information, designing visual tools to support and ameliorate the analysis of large multilayer graphs is essential. Recently, researchers from the *InfoVis* community have started to propose visualization techniques to deal with multilayer graphs. For instance, they have introduced specific visual encoding [5], new exploration systems [6], dedicated layout algorithms [7], etc. An overview of the state of the art is available in [8].

Designing visual graph query systems is also an active area of research [9], [10], [11], [12], [13], [14]. Considering a general visual graph query system, we can highlight four important tasks that must be managed: *Query Construction*, *Visualization and Exploration of Query Results*, and *Query Suggestion*. While researches mainly focus on tasks such as *Query Construction* and *Results Visualization* [9], [10], [11], [12], [13], [14], less research effort was devoted to deal with *Results Exploration* [11], and *Query Suggestion* [14]. To the best of our knowledge, no visual graph query system deals with these four tasks simultaneously on standard and/or multilayer graphs.

In this paper, we propose *VERTIGo*[1], a new visual platform that supports the users to query large multilayer graphs, visualizes/explores retrieved results and suggests new query extensions based on the underlying multilayer graph structure and the current query results. *VERTIGo* provides new mechanisms to integrate and manage the four tasks previously listed in a single application. In this way, it enables users to obtain information on multilayer graphs in an integrated context. Coordinated views support exploration and navigation of a large set of retrieved results at different levels of granularity. The views allow users to explore the results in detail as well as observe where they are located w.r.t. the initial graph, thus making it possible to pinpoint the cluster they belong to. Additionally, our platform also supports the synergy between the user and the underlying query process. The user can start, pause, and resume the query engine with the possibility to navigate/explore partially collected results. Fig. 1 illustrates our system while a demonstration video is available[2]. We remind that a standard graph defined on a single type of edge can be seen as a particular case of multilayer graph. This means that *VERTIGo* can also deal with a standard (single-layer) large graph.

The rest of this paper is organized as follows. Sec. 2 summarizes and discusses the related work. The requirement analysis is presented in Sec. 3. Sec. 4 introduces the proposed techniques while Sec. 5 introduces the interaction between the different visual components involved in *VERTIGo*. Two examples of use are described in Sec. 6. A usability evaluation and a discussion

- *Erick Cuenca is with Yachay Tech University, Urcuquí, Ecuador. E-mail: ecuenca@yachaytech.edu.ec.*
- *Arnaud Sallaberry is with LIRMM, the AMIS research group of the Paul Valéry University of Montpellier, the University of Montpellier and the CNRS, France. E-mail: arnaud.sallaberry@lirmm.fr.*
- *Dino Ienco is with INRAE, France. E-mail: dino.ienco@inrae.fr.*
- *Pascal Poncelet is with LIRMM, the University of Montpellier and the CNRS, France. E-mail: pascal.poncelet@lirmm.fr.*

1. A preliminary version of our system was presented as a demo paper at SSDBM [15]. Here, we detail the methodology, add new functionalities, illustrate the application with new examples, and perform a user study.

2. Demo video: https://youtu.be/0aC6-8pW66Y (accessed on June 22, 2020)

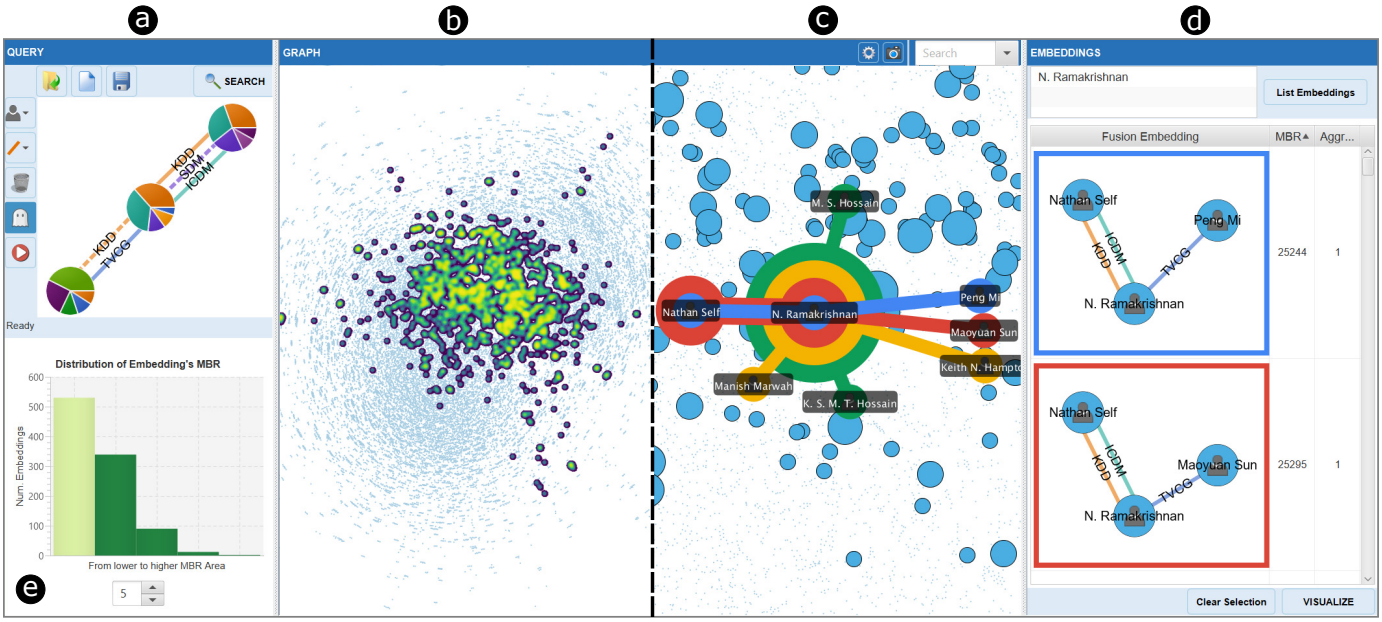


Fig. 1: The *VERTIGo*'s user interface being used to explore a co-authorship network. (a) The Query view allows the users to build the query and start the process to retrieve the results, called embeddings. This view also supports query suggestions visualized as pie charts. (b)(c) The Graph view showing either an overview of the embedding locations with a heatmap representation (b) or the embedding relations with Kelp-based diagrams (c). (d) The Embeddings view shows a list of embeddings for a set of selected entities (the embeddings involving N. Ramakrishnan in this example). (e) A histogram allows filtering embeddings by their Minimum Bounding Rectangle values.

of the advantages of our approach is presented in Sec. 7. Sec. 8 concludes.

2 Related Work

Gaining insights from large and complex graphs is challenging [1], [8], [16]. This requires the use of visual analytic tools with dedicated views, interactions, and techniques to assist users with exploration and navigation tasks. Pienta *et al.* [17] survey several visual graph systems and discuss the challenges related to making sense of large graph datasets. This includes the importance of combining graph visualization and exploration with data analytics methods to extract information. In this direction, the subgraph mining technique is valuable to help users discovering particular subgraphs (e. g., patterns) on a graph. This technique uses two approaches: i) *subgraph matching*, where all occurrences of a subgraph are searched on the target graph [9], [10], [11], [12], [13], and ii) *frequent subgraph mining*, where common subgraphs are identified in the structure of the graph [18], [19]. In this paper, we focus our study on the subgraph matching technique combined with visualization and interaction methods to retrieve results information from large multilayer graphs.

2.1 Multilayer Graph Exploration

By using different layers, multilayer graphs better model the complexity of various real world datasets (e. g., social networks, protein interactions network). However, using multiple layers on the same graph increases cognitive load when users interact with them [20]. Several approaches and techniques have been proposed to deal with the characteristics of such rich graphs. For instance, OntoVis [21] analyzes relationships in large heterogeneous social networks through the use of semantic and structural abstractions. It guides the analysis using filtering techniques and an ontology graph that describes relationships in the initial graph. McGee *et al.* [16] propose an approach to improve the exploration task on large multilayer graphs by extracting small set of data from the initial graph. This system provides a visual query builder interface that guides users through the relationships of the initial graph, allowing a subset of data to be built on demand. In their approach, the data is stored in a graph database (composed by a collection of graphs).

2.2 Visual Graph Query Systems

Given a query, find all its occurrences in a graph is computationally demanding, as it involves solving the subgraph isomorphism problem [22]. To overcome this problem, different subgraph matching algorithms [22], [23], [24], [25], [26] have been proposed. Several visual graph query systems use these algorithms as querying engines for matching subgraphs. GRAPHITE [9] proposes a visual interface to build a query on an author-publication network and display the results. To this purpose, it exploits the G-Ray [24] algorithm to retrieve subgraphs. VISAGE [10] provides a visual interface to guide the construction of the query. In this approach, the Neo4j[3] platform is employed to manage the graph database. In VIGOR [11], the query is firstly expressed by using the Cypher[4] query language and then transformed to a visual query. VIGOR also uses the Neo4j to manage the graph database. GraphVista [12] leverages a streaming mechanism that does not wait for the end of the query process, but instead visualizes the results as soon as they are available. GraphVista employs the

3. https://neo4j.com/ (accessed on June 4, 2020)
4. https://neo4j.com/docs/developer-manual/current/cypher/ (accessed on June 4, 2020)

Graphite [27] engine to retrieve subgraph results. A more sophisticated approach is proposed by VOGUE [13] where graph mining (gSpan [25]) and subgraph isomorphism (VF2 [23]) algorithms are combined to guide the user query process. Unfortunately, this approach works for transactional graphs settings where the graph database is constituted of a collection of graphs (e. g., a database of chemical molecules) and cannot be extended to work on a standard single large graph (e. g., social network, RDF data or biological network) where all the nodes of the graph database belong to the same big graph.

To summarize, the mentioned approaches working on single large graph provide limited interaction mechanisms among the user, the visual query system, and the query engine.

2.3 Visual Graph Query Suggestions

All the previous approaches mainly focus on two aspects: i) visually build the query and ii) display the results. Very few efforts have been done to reuse previous results to refine the querying process by suggestion [14], [28], [29]. VIIQ [14] proposes a visual interface to suggest the top-*k* relevant edges to extend the query by using user logs. It randomly samples a subset of the user logs then suggests actions that can match with the current query structure. The suggestions are not visually integrated into the query construction visualization, but are rather supplied with an additional panel showing the top-*k* results in terms of possible nodes/edges extensions.

Other approaches such as [28], [29] propose query suggestions using a list of possible nodes/edges to add to the current query; however, these approaches are dedicated to transactional graphs.

2.4 Visual Graph Query Processing Task

Table 1 shows a comparison of different visual graph query systems considering common tasks usually involved in the visual query process: *Query Construction*, *Visualization and Exploration of Query Results*, and *Query Suggestion*. We can note that, generally, the first two tasks are supported by all the systems. GRAPHITE [9], VOGUE [13], VIIQ [14], and VISAGE [10] use standard interaction techniques to visually draw the query structure (nodes, edges, and attributes) in a dedicated view and successively visualize query results. Conversely, VIGOR [11] does not provide a visual construction mechanism to build the query but it leverages the Cypher language to build it. GraphVista [12] allows the user to build the query via a node-attribute list, rather than visually drawing a query structure. Considering the *Exploration of Results* task, only the VIGOR [11] system provides mechanisms to browse results via coordinated views. Regarding the *Query Suggestion* task, VIIQ [14] is the only tool that supplies suggestion based on previous user actions. To summarize, most of the approaches focus on tasks such as *Query Construction* and *Visualization of Results* rather than *Exploration of Results* and *Query Suggestion*. To the best of our knowledge, there are no visual systems dealing with the four tasks simultaneously. In addition, the previous approaches only consider traditional graphs.

In this work, we introduce *VERTIGo*, a visual graph query system especially tailored for multilayer graphs. The proposed system introduces new mechanisms to cope with the query refinement process, the exploration/navigation of the retrieved results, and suggestions based on the graph structure.

TABLE 1: Comparison of various visual graph query systems to support query process tasks.

Approaches	Query const.	Visualization of results	Exploration of results	Query sugg.
GRAPHITE [9]	x	x		
VOGUE [13]	x	x		
GraphVista [12]	x	x		
VISAGE [10]	x	x		
VIGOR [11]	x	x	x	
VIIQ [14]	x			x
VERTIGo	x	x	x	x

3 Requirement Analysis

In this section, we describe the requirements of *VERTIGo* to effectively assist users in the process of querying a graph. Through an illustrative scenario, we conceptualize the main goals of our approach. In this example, we use a co-authorship network in the Visualization and the Data Mining fields of research. In this scenario, a user is interested in getting knowledge about authors who have published in the two fields of research. He or she starts with a query, in which the goal is to retrieve groups of authors that collaborate together and one of them has publications in both fields: he/she published a TVCG paper with one author and he/she also published an ICDM and a KDD paper with another author. This kind of query will give the user a guideline of the authors that is used as a link between the two communities. The user starts the query process on the graph. Once the results (groups of authors) are fetched, the user expects to see an overview of them. From this overview, the user can recover in which main community these authors are located, according to the co-authorship network, and thus obtain a starting point for a further analysis. In the detailed analysis, the user would be interested to know: a) the author with more occurrences in the aforementioned query, b) all the occurrences where an author is present, c) if two given authors are connected in an occurrence, and d) the locations of these authors regarding the co-authorship network. Based on this information, he/she would refine their previous query. For example, is there any other venue (e. g., *InfoVis*, *Sigmod*, and so forth) that was not taken into account in the initial query? From this scenario, we identified the following list of requirements:

[R1] *Graph query design and matching:* This requirement aims to provide the user with a tool to interactively design/construct a query. This query will serve as a starting point to find all the occurrences on the graph. In the previous example, the user can easily depict the authors and the conferences linking these authors with various venue types as journals or conferences (e. g. TVCG, ICDM, and KDD).

[R2] *Navigation and exploration of the results:* This requirement aims to provide visualizations and interactions to facilitate the exploration of the results, or the embeddings[5] at different granularity levels.

[R3] *Handling multilayer graphs:* This requirement aims to allow the user to model and depict a traditional graph or a multilayer graph, i. e., a graph where each layer contains edges of a certain type. Referring to the previous example, a layer can be a specific venue in the co-authorship network.

5. The term *embedding* refers to a subgraph matching the query structure. In this paper, it is employed as a synonym of *query result*.

[R4] *Query suggestion:* This requirement aims to suggest an extension of the initial query to the user. Mentioning the previous example, it could propose another conference venue based on the graph structure that links two authors in order to refine the results.

[R5] *Scalability:* This requirement aims to handle querying on large graphs and recover the embeddings in a reasonable System Response Time (SRT). SRT is defined as the time taken by the matching engine to evaluate the entire query.

The above list of requirements is intended to support frequent tasks involved in the visual graph query process, such as query construction, visualization of results, exploration of results, and query suggestions. Previous works [9], [10], [11], [12], [13] mainly cover the first two tasks: query construction and visualization of the results. However, to gain a better understanding of large and complex graphs, new visual approaches capable of taking advantage of the characteristics of multilayer graphs are needed [17], [20]. Therefore, this work aims to propose a tool to support the four important tasks involved in the process of graph querying. Since graphs are ubiquitous structures that can model data coming from many different domains [1], [8], our tool can be deployed in a wide range of scenarios. The previous list of requirements derived from an illustrative scenario intends to contextualize the benefits of a tool that can visually handle the graph query process.

4 Proposed Technique

This section describes our visual system. We first give a summary of the design rationale. We then detail each view. We leave the interactive functionality for the next section.

4.1 Design Rationale Summary

Based on the requirement analysis, we present *VERTIGo* (Fig. 1). It is based on three main visual components. The *Query* view (Fig. 1a) allows the users to visually construct/suggest the query structure (nodes, edges, node/edge-attributes) and start the graph engine processing. The *Graph* view (Fig. 1(b, c)) depicts the graph structure and the embeddings at different granularity levels. The *Embeddings* view (Fig. 1d) depicts in detail the structure of the embeddings for a given set of nodes. Below, we present these visual components and explain how they support the querying process.

4.2 The Query View

Fig. 2 shows the query view interface. This view allows the user to define the query structure (nodes, edges node/edge-attributes) via interactive functionalities **[R1, R3]**. This interface contains a drawing canvas, two built-in toolbars and a status bar. The drawing canvas (Fig. 2a) is the area where the query is constructed and where interactions happen. The design toolbar (Fig. 2b) provides different actions to build the query. From top to bottom, these actions are: add a node of a specific layer, add an edge of a specific layer, delete a node/edge, launch the edge suggestion mechanism and arrange the layout of the query by applying a force directed algorithm. The standard toolbar (Fig. 2c) includes frequently used features such as load, create, and save a query structure. It also contains the *Search* button that runs the graph engine. Finally, the status bar (Fig. 2d) provides textual information about the current state of the view.

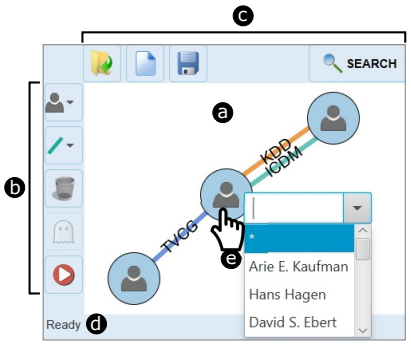


Fig. 2: The Query view displaying an example of a multilayer query. (a) The query is depicted in the drawing canvas. (b) The design toolbar provides interactions to construct the query. (c) The standard toolbar provides common features and also contains the *Search* button that starts the graph engine process. (d) A status bar shows textual information about the view. (e) The user can specify an attribute value on a node.

Actions such as specifying a node type or an attribute value, specifying an edge type, starting the query matching process and launching the edge suggestion mechanism require specific interactions described below.

4.2.1 Specifying a Node Type (Layer) and a Node Attribute Value

VERTIGo allows the users to specify the types of nodes in the query, which fit the multilayer aspect of the input graph **[R3]**. In this case, the list of matched embeddings only contains the nodes holding the selected types, and the SRT of the query process decreases **[R5]**. Nodes of a given type are represented by the same icon showing the type semantic (for instance, a person in Fig. 2e).

A specific attribute value can also be added to a node of the query in order to decrease the number of embeddings and the SRT of the query process **[R5]**. The query view enables this action by right-clicking on the desired node and selecting a value from the displayed pop-up list. For instance, in Fig. 2e, one can select an author name in a co-authorship network.

4.2.2 Specifying an Edge Type (Layer)

VERTIGo allows the users to query traditional graphs or multilayer graphs **[R3]**. We adopt a well-known way to visualize the edge types (or layers) by color coding. When a pair of nodes is connected by several edges, some scalability challenges arise i) how to organize these links and ii) how to deal with many edge types. To overcome these problems, edges between a pair of nodes are arranged in a parallel way. The relative distance between parallel edges remains the same even when the number of links increases, which allows the users to clearly see all the links, as shown in Fig. 2. The upper limit for the number of edges is 10 because of the known problem of distinguishing small regions of colors [30]. Edges are labeled to enhance the type recognition. In order to add an edge to a query, the user must first choose an edge type from the drop-down edges button on the design toolbar (Fig. 2b). Then, using the mouse pointer, the user draws the edge from one node to another. Finally, the edge is automatically arranged. When more than two edges connect a pair of nodes, the Boolean logical operator *AND* is used to perform a logical conjunction between them.

Fig. 2 shows the query example described in Sec. 3, where an author published a TVCG paper with one author and he/she also published an ICDM and a KDD paper with another author. More precisely, we explicitly state that the author corresponding to the central node has published both an ICDM and a KDD article with the same co-author, and a TVCG paper with another co-author. We can observe that the multilayer graph structure allows us to easily specify multiple constraints between the same pair of nodes.

4.2.3 Starting the Query Engine Process

Since a multilayer graph involves different nodes and edge types, standard query engines such as TurboISO [22] or VF3 [26] are not adapted for such graphs. To deal with multilayer graphs **[R3]**, we integrate into *VERTIGo* a recent multilayer graph query engine named *SuMGra* [31]. It exploits specialized indexing techniques to speed the backtracking algorithm that is commonly employed to deal with the subgraph isomorphism problem [32]. The *Search* button in the standard toolbar (Fig. 2c) starts the query process.

A common paradigm in query process is based on a *Query→Result* approach, i. e., after a query construction, send the request to the query engine and then visualize the retrieved results [12]. However, as interactions with the query engine are not available in this paradigm, the process can be affected by huge SRT when the number of results grows exponentially. In order to improve the SRT **[R5]**, *VERTIGo* tightly interacts with the SuMGra system to enable the user to start, pause, and resume the query engine with the possibility to navigate/explore partially collected embeddings **[R2]** (this functionality is described in Sec. 5).

4.2.4 Visualizing Query Suggestions

Based on the initial query structure, *VERTIGo* automatically suggests k edges to the user's query intention **[R4]**. To obtain *candidate edges*, VIIQ [14] uses a correlation of a subset of edges of a user log. This approach only ranks a subset of edges which leads to suggest only a part of the query structure. *VERTIGo* overcomes this drawback by ranking all the adjacent edges for every node of the embeddings and then suggests the top-k frequent type of edges found. The graph structure is used to obtain the adjacent edges for each embedding node. This mechanism suggests interesting extensions of the query since it leverages information obtained by the underlying graph.

VERTIGo suggests two types of candidate edges: *internal edges* that link nodes that are already in the query, and *external edges* that link nodes of the query to other ones. Internal and external suggested edges are displayed directly on the query. The left of Fig. 3 shows the query of Fig. 2 including visual suggestions. Internal edges are represented by dashed lines to differentiate them from existing edges (Fig. 3(a, b) *KDD* and *SDM* links). Since external edges convey extension to an external node, we use a pie chart. The number of slices in the pie chart is equal to k (the number of external edge suggestions Fig. 3c). The percentage represented by a slice and displayed on its tooltip depicts the quantity of edges in the neighborhood of the embeddings that support the extension of the query by the particular edge type. For instance, Fig. 3c indicates that 10% of the embeddings already found can be extended with a *CGF* edge type.

The right part of Fig. 3 shows the query after adding the above-mentioned suggestions. To accept a suggested edge, the user clicks on the internal (dashed line) or external edge (pie slice). A new internal edge is displayed as a continuous line (Fig. 3(d, e) *KDD* and *SDM* links). A new external edge is displayed as a link to a node toward its slice (Fig. 3f *CGF* link). The query suggestion mechanism provides visual elements to guide the user in the incremental construction of the query **[R1, R4]**. Thus, each time the query is extended, the user can retrieve embeddings and further refine his/her searching process. The query suggestion mechanism is activated from the design toolbar (Fig. 2b).

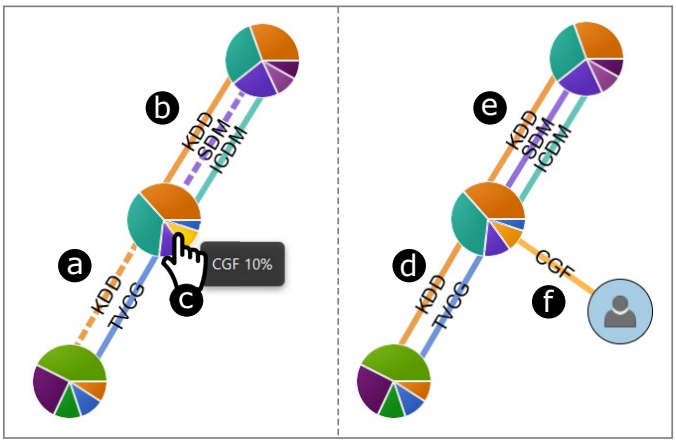


Fig. 3: An example of a query suggestion. On the left, the visual elements on the query depict suggested edges. On the right, the query structure includes newly added suggested edges.

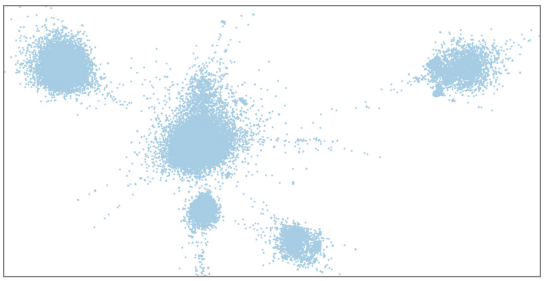
4.3 The Graph View

We adopt a node-link diagram with a force directed layout [33] to produce a suitable visualization of the graph. Working with large graphs entails some challenges: aesthetic criteria, scalability and reasonable interaction time. To deal with these issues, we leverage a multilevel technique named Multipole Multilevel Method (FM^3) [34]. FM^3 is a well-known approach in the graph drawing field that allows scaling up to large graphs with a good trade off between time and visualization performances **[R2, R5]**. An example of a graph layout obtained by *VERTIGo* is presented in Fig. 4a. The input graph is a biological dataset [3] modeled as a multilayer graph. It is composed of 38,936 nodes, 310,664 edges and 7 edge types. We note that such a visualization has the advantage to stand out groups of nodes that clearly form communities. In this particular example, we can easily highlight five major groups (clusters) of nodes. Edges of the graph layout are not depicted to avoid cluttering issues.

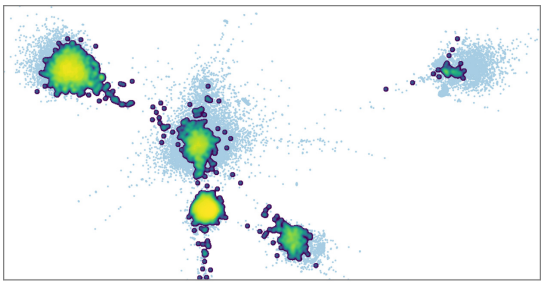
4.4 The Embeddings View

In order to display a large number of embeddings, Pienta *et al.* [11] reduced them to points. This kind of abstraction avoids a cluttered visualization; however, the spatial distribution of its elements is not depicted. With the aim to display in detail the topology of the embeddings (i. e., nodes and edges) **[R2]**, we designed the Embeddings view (Fig. 5). In this view, we show in detail the structure of the embeddings (Fig. 5b) containing a given set of nodes (Fig. 5a) (interactions to select these nodes are described in Sec. 5).

The goal of the Embeddings view is to allow the user to sample the embeddings for any combination of nodes involved in the embeddings. For example, considering the embeddings of the query shown in Fig. 2, the Fig. 5b shows the list of embeddings



(a)



(b)

Fig. 4: The Graph view displaying a biological dataset. (a) The obtained layout allows the users to easily identify five different communities. (b) A heatmap shows an overview of the retrieved embedding locations in the graph.

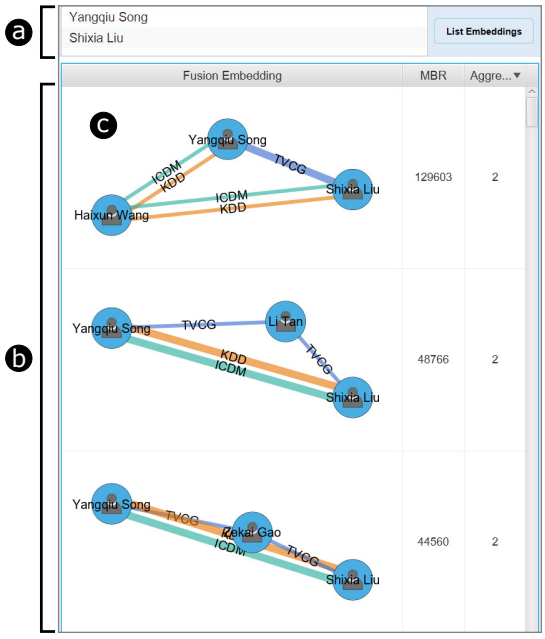


Fig. 5: The Embeddings view showing a list of embeddings (b) for a set of entities (a). (c) An example of a *fusion embedding* where the thickness of the edges conveys the number of aggregations.

where *Shixia Liu* and *Yangqiu Song* are present. This behaviour enhances the exploration task **[R2]** since it not only shows the embeddings, but also a subset of them.

In order to convey the spatial distribution of the embeddings, we use the Minimum Bounding Rectangle (MBR) value that bounds the nodes of an embedding in the Graph view. This value transmits the extent of the nodes within an embedding. For instance, a low MBR value conveys the closeness of its nodes, while a high MBR value means that one or more nodes are far apart from the others, possibly in different clusters. The Embeddings view allows users to sort results in an ascending/descending order based on their MBR values **[R2]**. This metric is also used to filter results using a histogram (Fig. 1e) that groups MBR values into n ranges (interactions are described in Sec. 5.2).

With the aim to depict the topological structure of the embeddings, we must consider that some embeddings could involve the same set of nodes, with a different edge topology. Based on this feature, we aggregate embeddings with the same set of nodes into a single *fusion embedding*. Fig. 6 shows an example where the fusion embedding (c) is the result of the fusion of the embeddings (a) and (b). Thus, this fusion maintains the topology of nodes, but the topology of edges evolves according to the links in their aggregated embeddings. The number of aggregations is represented by the thickness of the edges in Fig. 6c.

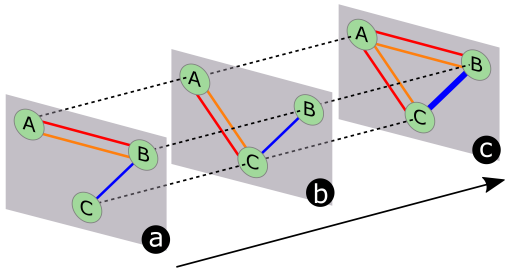


Fig. 6: Construction of the fusion embedding (c) from the fusion of embeddings (a) and (b). The topology of the nodes is maintained while the topology of the edges evolves. The thickness of the edges reveals the number of aggregations.

Each row in the list of embeddings (Fig. 5b) is composed by a fusion embedding and its associated metadata (i. e., the MBR value and the number of aggregated embeddings). This metadata can be particularly useful to explore embeddings according to their spatial/topological features. For instance, in a co-authorship network, given a set of authors (Fig. 5a), if the user is interested in knowing other authors who often collaborate with them, he/she can order up the list of embeddings (Fig. 5b) by clicking on the last column of the view, and recover the embedding(s) with the highest number of aggregations (Fig. 5c).

In this section, we have presented the three main visual components of *VERTIGo*: the *Query* view, the *Graph* view, and the *Embeddings* view. Each view performs a specific task on the visual query process. The *Query* view allows the visual query construction and query suggestion. The *Graph* view shows the graph structure and allows users to navigate/explore embeddings at different granularity levels. The *Embeddings* view allows the users to visualize a list of embeddings for further investigations. In the next section, we describe the interactions among these visual components.

5 Interactions

In this section, we describe the interactions between the visual components involved in *VERTIGo*. Interactions are tailored to help the user on the visual query process: query construction, results visualization, results exploration, and query suggestions. Fig. 7 shows the general architecture of *VERTIGo*, where arrows represent the interactions among the components. The user and system operations are represented in blue and green respectively.

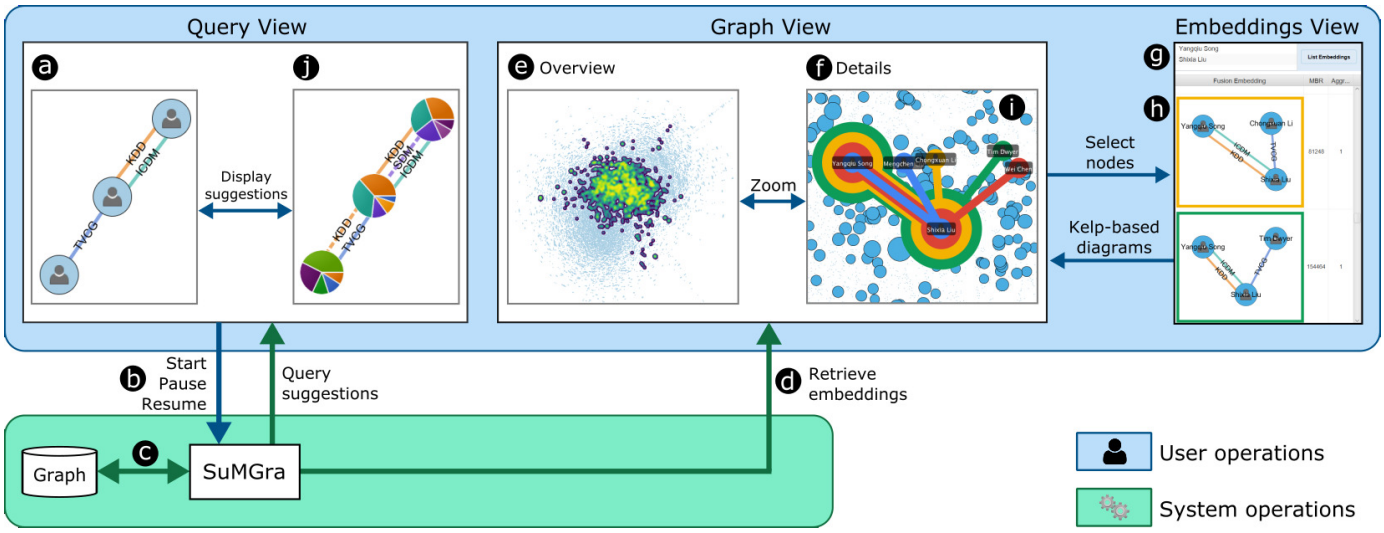


Fig. 7: System architecture of *VERTIGo*. Colored arrows show interactions between components. User operations are represented with blue arrows, while system actions are shown in green.

5.1 Visualizing Embeddings

The user starts the visual query process by building the query structure (Fig. 7a). Once the construction of the query is finished, the user sends the query to the engine (Fig. 7b). Next, the query engine requests the graph structure in order to retrieve the embeddings (Fig. 7(c, d)) and returns them to the Graph view. Based on the retrieved embeddings and the graph structure, the visual query mechanism suggests k edges to refine the previous query and execute it again (Fig. 7j).

Since the number of embeddings can be large, the visualization is challenging. To overcome this issue, *VERTIGo* enables two different levels of analysis: i) an overview level in which it sums up the embeddings according to the graph (Fig. 7e) and ii) a detailed level in which it allows the user to inspect particular nodes involved in the results (Fig. 7f). The user navigates between the overview and the detailed levels performing a semantic zooming (i. e., changing the detail levels of the graph and the embeddings).

5.1.1 Overview Level

Considering the overview, we employ a heatmap [35] that dynamically illustrates the portion of the graph in which embeddings are located. Fig. 4b shows an example of a heatmap of embeddings. The density distribution is represented using the viridis colour gradient [36] which offers a high range of perceived values while avoiding red-green color blindness. Liu and Heer [37] demonstrate that this gradient performs well to encode quantitative data in terms of time and error through an experimentation involving 9 competitors. The viridis scheme varies in a sequential multi-hue palette from blue (low-dense areas) to yellow (high-dense areas). In the example depicted in Fig. 4b, we can highlight that the Graph view shows different communities since the layout algorithm highlights different groups of nodes. Here, the embeddings are not equally distributed among these groups. As an interesting point, we can note a high density (high concentration of embeddings depicted in yellow) in the center of the image while a lower concentration of embeddings appears in the upper-right corner. Our heatmap can be employed to visually understand if a query is specific to a certain portion of the graph or not, and thus provides the user an entry point for the exploration task **[R2]**.

We would like to note that *VERTIGo* allows the user to pause and resume the underlying query process (Fig. 7b). This can be particularly useful if the number of embeddings grows quickly **[R5]**. In that scenario, the process can be paused at any moment, the embeddings are retrieved considering the current state of the query engine and the corresponding heatmap (Fig. 7e) is produced or updated. Finally, the user can resume the query process from where he/she had stopped.

5.1.2 Detailed Level

The abstraction supplied by the heatmap makes it difficult to know the exact number/distribution of embeddings found in a region. To deal with this issue, *VERTIGo* allows the users to change the levels of details zooming on some particular area **[R2]** (Fig. 7f). In this case, the nodes of the embeddings are shown in a contrasted color (see the blue nodes in Fig. 8). We use the area of the node to depict the number of embeddings it belongs to: bigger nodes correspond to nodes occurring in more embeddings. In order to enhance the effectiveness of the exploration task **[R2]**, we show the labels of embedding nodes. Labeling the nodes is not a trivial task, since showing labels for all nodes may cause cluttering issues. To overcome this problem, we show overlapping-free labels based on node weight (i. e., the number of times it appears in an embedding), by using a greedy algorithm. It first orders the set of nodes by weight. Next, for each node, it calculates the bounding rectangle of the label, and it checks if there is no overlap with the previous labeled nodes. If there is no overlap, the label is shown, otherwise it is not. The complexity of this algorithm is $O(n^2)$, where n is the set of embedding nodes. The result depends on the zoom level. Thus, when the user zooms in/out, the algorithm computes again the displayed labels. Fig. 8 shows an example. Notice that some node labels are bigger (e. g., *N. Ramakrishnan*) than their neighbors.

The labeling algorithm clearly displays the most weighted node labels. However, depending on the zoom level, some nodes with low weight may be underneath others. To overcome this problem, we implement a text field search facility to retrieve nodes by their name. Fig. 8a shows this option. Once the node name is

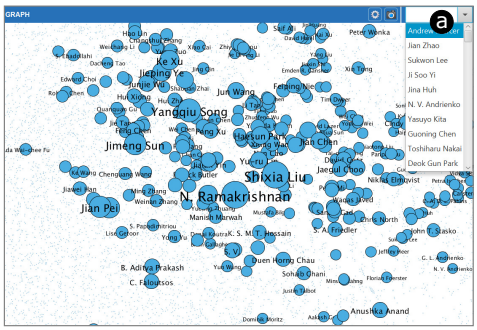


Fig. 8: The Graph view showing the nodes of the retrieved embeddings. The area of a node represents the number of embeddings it belongs to. (a) A text field allows the search for a specific node by its name.

selected, the focus of the Graph view is automatically redirected to the node location and an animation shows its exact location.

From the detailed level, the user can select a set of embedding nodes for further analysis by clicking on them. Their border color becomes red and they are automatically added to the Embeddings view (Fig. 7g). The user can thus visualize the list of the associated embeddings (Fig. 7h). In the following section, we describe the exploration of embeddings using both: the Embeddings and the Graph views.

5.2 Exploring Embeddings

The Embeddings view displays the list of fusion embeddings (Fig. 7h) containing the set of the selected nodes (Fig. 7g). This list allows the user to explore embeddings based on their spatial (MBR values) and topological (aggregations) structure; however, the specific location on the graph is not shown. Since the whole graph is shown in the Graph view (Fig. 7(e, f)), we aim to display embedding locations in this view. Considering the presence of nodes in multiple embeddings, several existing methods such as BubbleSets [38], LineSets [39] and Kelp diagrams [40] seem suitable. All these approaches form an appropriated visualization when the positions of nodes are fixed. Among all these approaches, we chose in our work the Kelp diagrams due to their ability to emphasize aesthetic criteria to deal with overlapping in set membership. Kelp diagrams act on static layouts (such as maps) without user interactions. Since the Graph view (Fig. 7(e, f)) provides a zooming technique that changes node positions, we must adapt the Kelp diagrams to our needs. Our Kelp-based approach is described in Sec. 5.3.

From the list of embeddings (Fig. 7h), the user can select different fusion embeddings that will be visualized on the Graph view by using a Kelp-based approach (Fig. 7i). When the user selects a fusion embedding, the row border is colored according to a categorical color scheme. Ware [30] mentions that there are up to seven user-distinguishable colors. In our implementation, we use a five color scale (red, green, blue, orange, and pink) and we can then go up to five colors. If we go beyond five, we return to the same colors. The same color is used in the respective Kelp-based diagram on the detailed level of the Graph view (Fig. 7i).

Besides the Kelp-based representation, *VERTIGo* provides a histogram to explore embedding results based on their MBR value **[R2]**. MBR values are grouped into n ranges defined by the user, where $1 \leq n \leq 10$. Fig. 1e displays a histogram grouped into five ranges. In this example, we can highlight that there is a concentration of embeddings with a low MBR value, which reflects their proximity in the graph (maybe in the same cluster). On the other hand, the last two ranges show a low concentration of embeddings with a high MBR value (nodes of these embeddings might be in different clusters). This histogram allows the user to filter embeddings by clicking over the desired bar. Thank to this functionality, users can focus on a specific group or discover anomalous embeddings (please refer to the example on Sec. 6.1 to see this functionality).

5.3 The Kelp-based Approach

When zooming in the Graph view, the positions of the nodes change according to the focal point (guided by the mouse pointer). Thus, at a certain zoom level, some nodes can be affected by nodes and/or edges overlap. Unfortunately, even if Kelp diagrams are very attractive for highlighting embeddings, they are not suited to deal with non-predefined node positions. Their high computational burden negatively influences their use to manage real-time interaction. To deal with this issue, we extend the standard Kelp technique with a strategy to manage non-predefined node positions. Our Kelp-based approach has two main parts: i) find a spatial arrangement of nodes at the current zoom level and ii) generate a nested visualization of them.

5.3.1 *Positioning the Nodes*

The following pseudo-code presents the steps of our approach to position the nodes in order to generate an overlapping free diagram. We consider each embedding as a *subgraph* E of the initial graph. $\mathbb{E}$ denotes the set of the embeddings and $\mathbb{N}$ denotes the set of nodes in $\mathbb{E}$.

1: For every $n \in \mathbb{N}$, set a circle of radius r_n corresponding to the number of embeddings of $\mathbb{E}$ containing n (Fig. 9a).
2: Derive a Delaunay triangulation of the set of nodes $\mathbb{N}$ and for each edge of the Delaunay triangulation, eliminate the overlapping employing a stress model [41] (Fig. 9b).
3: Let $\mathbb{L}$ be the set of links to draw between the nodes of $\mathbb{N}$. Thus, $\mathbb{L}$ contains the union of the edges of the embeddings $\mathbb{E}$. For every link $l \in \mathbb{L}$, the thickness r_l corresponds to the number of embeddings of $\mathbb{E}$ containing l (Fig. 9c, in which node-link overlaps appear).
4: For every node-link overlap, set a *fixed-control-node* $c \in \mathbb{C}$ at the barycenter of the overlapping region with a radius corresponding to the distance from the barycenter to the border of the link (Fig. 9d). Then, add the set $\mathbb{C}$ to $\mathbb{N}$.
5: Re-execute the node overlapping process described in step 2 over the current set of nodes $\mathbb{N}$, the set of $\mathbb{C}$ remains fixed over this process (Fig. 9e). Once finished, remove the set of $\mathbb{C}$ from $\mathbb{N}$.

We now give the details of each step.

Step 1: The first step of our approach aims to locate the set of nodes $\mathbb{N}$ and the set of embeddings $\mathbb{E}$ avoiding overlap at the current zoom level. We display every node $n \in \mathbb{N}$ as a circle of radius r_n ($r_n \geq 1$) corresponding to the number of embeddings $E \in \mathbb{E}$ to which it belongs.

Step 2: Considering the circles and the positions of the nodes at the current zoom level, some of them may overlap. To deal with this problem, we remove node overlapping by using the approach described in [41] (see [42] for an introduction to node overlap removal algorithms). The approach proposes to derive the

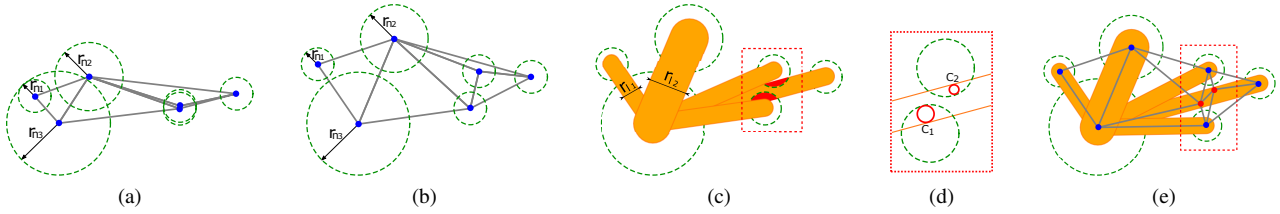


Fig. 9: Node-link overlapping process: (a) Detection of possible node overlapping (b) Removing node overlapping using a Delaunay triangulation approach (c) Adding links between nodes (orange) and detection of possible node-link overlapping (red box) (d) Determination of position of fixed-control-nodes (in red) (e) Fixed-control-nodes are used to re-calculate the node-link overlap process into the layout.

Delaunay triangulation from the overlapping layout (where the radius sizes of the nodes are greater than zero) and then, for each edge of the triangulation, to gradually eliminate the overlapping employing a stress model. Following this approach, the Delaunay triangulation from the set of nodes $\mathbb{N}$ is derived (Fig. 9a) and the stress model (Fig. 9b) is successively applied. The result of this method is an overlapping-free graph that allows the users to preserve the mental map of the initial layout. Furthermore, the method also maintains a reasonable computational complexity to facilitate real-time interaction.

Step 3: After removing nodes overlapping, we aim to visually connect the nodes in every embedding $E \in \mathbb{E}$. We define $\mathbb{L}$ as the set of links to draw between the nodes of $\mathbb{N}$. Each link $l \in \mathbb{L}$ holds a thickness r_l that is equal to the number of embeddings of $\mathbb{E}$ containing l. Fig. 9c shows these links in orange. After this step, some links can overlap with the nodes, (see the red boxes in Fig. 9c).

Step 4: To solve the issue related to possible node-link overlaps, we introduce a set of *fixed-control-nodes* ($\mathbb{C}$) around the barycenter of the overlap regions. The radius is given by the distance from the barycenter to the corresponding overlapping link l. Then, the set of the *fixed-control-nodes* $\mathbb{C}$ is added to the set of nodes $\mathbb{N}$. Fig. 9d shows two control nodes, c1 and c2 on the overlapping regions depicted in Fig. 9c.

Step 5: finally, we perform again the node overlapping process described in step 2 including the augmented set of control nodes $\mathbb{C}$ to $\mathbb{N}$. The position of the control nodes remains fixed during the node-overlapping elimination process. Thus, link positions are preserved and only overlapping nodes are rearranged. Once this process is finished, fixed-control-nodes $\mathbb{C}$ are removed from the set of nodes $\mathbb{N}$. Fig. 9e shows the final result of this process, displaying the Delaunay triangulation in gray and the barycenter of the control nodes in red. Once we obtain a suitable allocation of both nodes and links, we draw them.

5.3.2 Generation of a Nested Visualization

When nodes are positioned, we need to visualize the embeddings as nested sets. We provide two different styles: a Kelp-based diagram without edges and another one with edges (Fig. 10(a, b)). To generate these styles, every embedding is assigned to a colored layer in a stacked structure. Nodes and edges are stacked on top of each other modifying the radius of the nodes and the thickness of the edges to make them visible. The color of an embedding is assigned from the border row of the corresponding embedding E in the Embeddings view (Fig. 7h).

Fig. 10 shows the nested visualization with hidden edges (a) and visible edges (b). In this example, the Kelp-based diagrams show four selected embeddings. We note that, when nodes or edges are involved in many embeddings they become visually dominant (e. g., nodes for *Yangqiu Song* and *Shixia Liu* authors). We label nodes and edges in order to facilitate their identification and enhance exploration tasks (Fig. 10c). Labels of the edges are shown when the user hovers the mouse on an embedding. The selected embedding is also highlighted while the others decrease in opacity. In addition, *VERTIGo* provides a user control panel to change from one nested style to another one on the fly, or modifying some visual parameters such as the initial radius of nodes, the thickness of links, the font label size, etc.

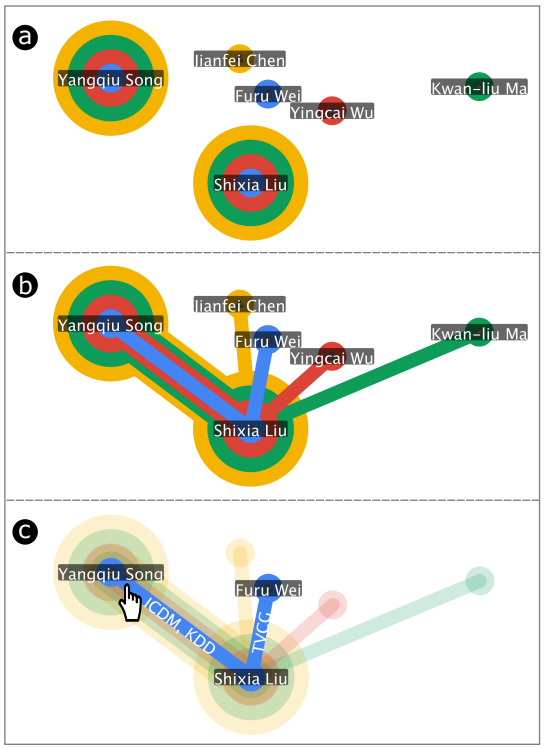


Fig. 10: Kelp-based nested styles: (a) links are not shown and (b) links are visible. (c) When hovering an embedding, it becomes highlighted and the edge labels are displayed.

5.4 Implementation and Equipment

VERTIGo is developed using Java technologies (JavaFX, Processing[6]) for its interface and interaction among the different com-

6. https://processing.org/ (accessed on June 4, 2020)

ponents. We use the FM^3 implementation supplied by OGDF[7]. The graph database engine [31] is also implemented in Java and integrated in the system. Demonstration scenario is carried out on a Work Station with an Intel Xeon processor at 3.00 GHz with 16 GB of RAM memory with a 64-bit Windows OS.

6 EXAMPLES

In this section, we present two examples illustrating how *VERTIGo* can be used to explore multilayer networks.

6.1 Co-authorship Network

In this section, we consider a real co-authorship network obtained from the *DBLP*[8] bibliographic database. The network is composed of 37,878 nodes (the authors) and 129,983 edges (co-authorship). The multilayer network has 18 layers (edge types), each of them corresponding to a different venue (journal or conference) in which authors have published. Two nodes are linked together if the corresponding authors have co-authored at least one paper together in a specific venue. The different venues are chosen considering the Visualization (VIS) and the Data Mining/Data Base (DMDB) fields of research. Regarding the VIS domain, we have selected the following venues: *TVCG*, *InfoVis*, *CGF*, *EuroVis*, *PacificVis*, *Graph Drawing*, *CG&A*, and *IV*. Considering the DMDB domain the chosen venues are: *DASFAA*, *EDBT*, *ICDE*, *ICDM*, *ICDT*, *KDD*, *SDM*, *SSDBM*, *SIGMOD*, and *VLDB*.

The bottom of Fig. 11a shows the DBLP multilayer graph displayed in the Graph view **[R3]**. The nodes (authors) are shown as blue circles of equal sizes, edges are not depicted to avoid cluttering. We can note that the chosen directed force layout algorithm allows the users to visually recognize two dense areas. In this example, we aim to retrieve the *co-authorship network* of a given author. More precisely, using the k suggested edges provided by the query suggestion mechanism, we want to visualize the network around *Yehuda Koren* **[R1, R2, R4]**. The top of Fig. 11a shows the initial query. The depicted graph involves two authors (nodes) that have co-authored a *TVCG* and a *Graph Drawing* paper. In addition, one of the two nodes is identified as *Yehuda Koren*. Once this query is executed, the Graph view displays the retrieved embedding results via a heatmap in order to locate them on the original graph (the bottom of Fig. 11a) **[R1, R2, R3]**. The heatmap shows a concentration of embedding results in the right center part of the DBLP network, which leads us to think that the dense area on the right corresponds to the VIS community.

With the aim to visually extend the *Koren's* collaboration network, we use the query suggestion mechanism provided by *VERTIGo* **[R2, R4]**. The top of Fig. 11b shows the k-edges suggested to extend the query. We focus on the external suggested edges (i. e., pie charts). The percentage displayed on the slice informs us that 41% of edges in the DBLP network support the extension of a new *TVCG* link to another co-author. Thus, we add this suggested edge to the query structure and we execute the query process again. The bottom of Fig. 11b shows the embedding results: they are concentrated on the right community. This further evidence confirms that the right area of the graph corresponds to the VIS community.

Once this community is detected in the network, we continue adding suggested edges with the objective to observe connections involving authors from the DMDB community **[R2, R4]**. The top of Fig. 11c shows a query structure where one of the top-5 suggested edges corresponds to the KDD conference. We add it to the query. The bottom of Fig. 11c shows the results. They start to spread toward the left part of the DBLP network. The top of Fig. 11d shows the query structure after adding an *ICDM* link to a new co-author. The embedding results show even more concentration of results into the left community, confirming that this part of the network represents the DMDB community.

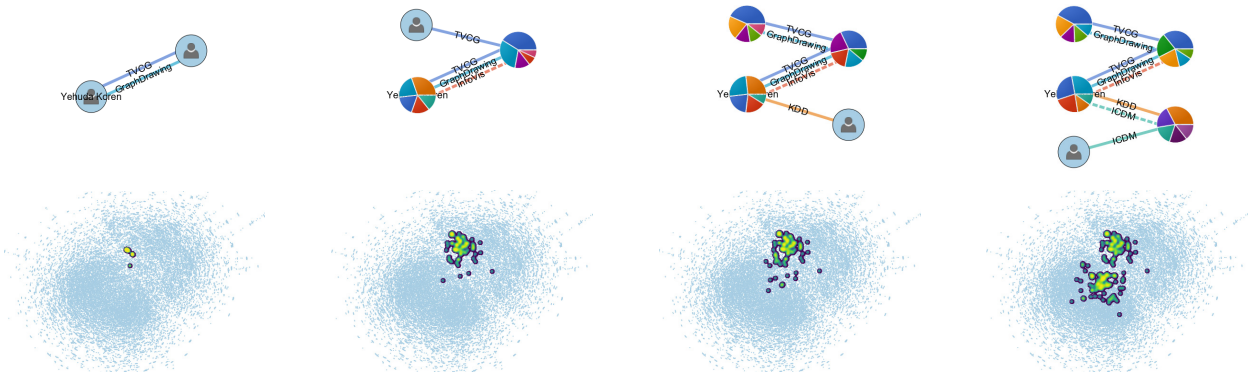
After four refinement steps achieved by adding suggested edges, we can infer that the Koren's co-authorship collaboration network involves many researchers from both communities VIS and DMDB. Moreover, as the suggestions on his node directly concern the two communities (*TVCG*, *Graph Drawing* and *KDD* developed in our example), we can assume that this author is a hub between the communities. This is confirmed by the whole list of publications from the author containing many VIS articles (*TVCG*, *Graph Drawing*, *InfoVis*, *etc.*), as well as many DMDB articles (*KDD*, *ICDM*, *TKDD*, *etc.*).

Guided by the heatmap of Fig. 11d, we zoom to the center of these two communities (see Fig. 12a). The font size for *Yehuda Koren* is bigger than the other node labels since this author appears in all the query results **[R2]**. He is located in the middle of both communities, which makes sense since he acts as a hub between them. In the histogram organized into 5 ranges, we can observe a lot of embeddings with a low MBR value, i. e. located in the same region of the graph. In Fig. 12b we filter the first four bins in order to concentrate the visual analysis on embeddings with a high MBR (the authors that can be in different communities). As a result, the Graph view shows fewer embedding nodes, which indicates less collaboration between central authors of the two communities. We then display the list of embeddings on the Embeddings View. From this list, we select three of them. Fig. 12c shows the corresponding Kelp-based diagram. *Yehuda Koren* appears in the center as a bridge between researchers who contribute primarily in their respective communities. For instance, *V. S. Subrahmanian* from DMDB and *D. Weiskopf* from VIS appear as a central nodes of their community, but less prone to collaborate with researchers from the other community. This is confirmed with their DBLP record in which their venues are mostly related to their own field of research.

6.2 Offshore Leaks Network

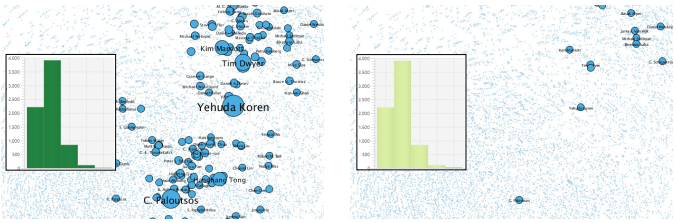
Our second example is based on a real offshore dataset revealed by the International Consortium of Investigative Journalists (ICIJ)[9]. This dataset exposes the relationship between entities (persons, companies, clients, and addresses) in tax heavens around the world through four investigations called the *Panama Papers*, the *Offshore Leaks*, the *Bahamas Leaks*, and the *Paradise Papers*. The structures in these investigations contain three possible types of relationship between entities: i) a person *isOfficer* of a company, ii) a person *hasRegistered* an address, and iii) a client *isIntermediary* of a company. We use this data to model our multilayer graph which is composed by 406,072 nodes (the entities) and 664,901 edges (the relationships) **[R3, R5]**. Our multilayer graph has 12 layers (edge types) that represent the types of relationship and the investigation to which it belongs, and thus we obtain the layers: *isOfficer_panama*, *isOfficer_offshore*, *isOfficer_bahamas*, *isOfficer_paradise*, *hasRegistered_panama*, *hasRegistered_offshore*,

7. http://www.ogdf.net/ (accessed on June 4, 2020)
8. http://dblp.uni-trier.de/ (accessed on June 4, 2020)
9. https://offshoreleaks.icij.org/pages/database (accessed on June 4, 2020)

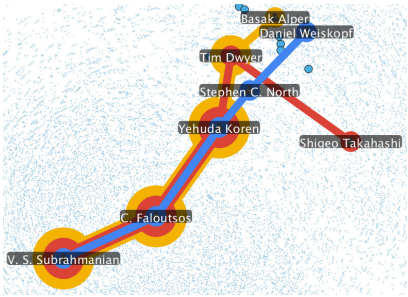


(a) (b) (c) (d)

Fig. 11: Query refinement process exploiting the suggestions provided by *VERTIGo*.



(a) (b)



(c)

Fig. 12: (a) Detailed level of the embedding results from the Fig. 11c. Note the histogram with a high concentration of embeddings with a low MBR value. (b) Embedding results after filtering embeddings with high MBR value. (c) The Kelp-based diagram associated to the selected embeddings.

hasRegistered_bahamas, *hasRegistered_paradise*, *isIntermediary_panama*, *isIntermediary_offshore*, *isIntermediary_bahamas*, and *isIntermediary_paradise*.

When analyzing the offshore dataset, an interesting starting point is a person sharing relationships with different companies across investigations **[R3]**. For instance, a person P who is an officer of a company C_1 in the *Panama Papers* and who also is an officer of another company C_2 on the *Offshore links* can be used to start further analysis of the relationship between these companies and the role of the person among them. *VERTIGo* can easily help to identify such people by launching the query shown in Fig. 13. In this example, the query contains three people, two of them being officers of a company in two distinct investigations (*Panama Papers* and *Offshore links*) and the third one being an officer of both companies **[R1, R3]**.

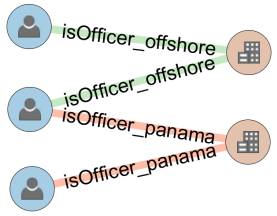
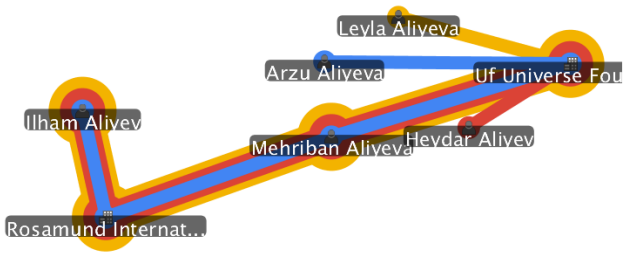


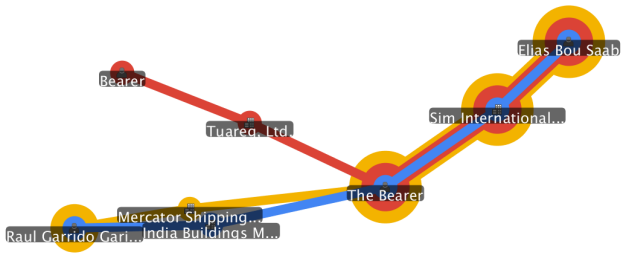
Fig. 13: Query to extract people who are officers of different companies in distinct investigations.

Once this query is executed and *VERTIGo* gets the results, we observe two sets of embeddings involving politicians **[R1, R2, R3]**. The first one is shown by Fig. 14a. We can observe three embeddings in blue, red and orange. *Ilham Aliyev*, President of Azerbaijan, belongs to these three embeddings, as does his wife, *Mehriban Aliyev*. She is the one who acts as a bridge between the two investigations and thus, the two companies (match with the central node of the query). She and President *Aliyev* are officers of the *Rosamund International* company in the *Offshore Leaks*. The President's daughters, *Leyda* and *Arzu*, and his father, *Heyda*, are officers of the same company, the *UF Universe Foundation*, in the *Panama Papers*. Further investigation on the family states their implication in a vast affair of hidden wealth [43]. While this example had been identified by the journalists of the ICIJ by manual inspection of the ten thousand entities contained in the dataset, we have demonstrated here how *VERTIGo* can support such investigation, automatically, reducing human effort and time.

The second set of embeddings involving a politician is shown in Fig. 14b. The Kelp-like diagram depicts three embeddings in which *Elias Bou Saab*, a Lebanese Minister is present. We see that a person labeled as *The Bearer* links both investigations (*Panama Papers* and *Offshore links*). This entity and Minister *Bou Saab* are officers of the *Sim International Services Limited* company through the *Panama Papers* investigation. By counting the number of circles that surround them, we can conclude that they participate in the three result embeddings while the other entities participate in fewer results **[R2]**. *The Bearer* is particularly interesting as it does not reveal its real name. The relationships he/she has with the other entities could be a starting point to support further investigations. Another question lies on the relation between *Bearer* and *The Bearer*: are they the same person?



(a)



(b)

Fig. 14: Sets of embeddings extracted with the query shown in Fig. 13. (a) Relationship of the family of the President of Azerbaijan, *Ilham Aliyev*. (b) Relationship of a Lebanese Minister, *Elias Bou Saab*.

7 Evaluation and Discussion

In this section, we discuss the usability, the usefulness and the performance of our work. We carry out a user study to obtain a qualitative evaluation of specialized users.

7.1 User Study

This study has two objectives. The first one is to introduce *VERTIGo*'s functionalities to potential users (e. g., investigative journalists), and the second one is to obtain feedback to evaluate the capabilities of our work in a practical context. The steps to achieve these goals are: i) choose an appropriate dataset, ii) determine domain experts to involve in the study, and iii) conduct a survey to measure the usability and effectiveness of *VERTIGo*. These activities are described below:

Choose a dataset: We chose the real dataset of the Offshore Leaks Networks described in Sec. 6.2. A suitable characteristic of this dataset is that it contains different types of nodes and edges in a large multilayer graph.

Determine domain experts: Even if no particular skills are needed to use our system, in the case of the Offshore Leaks Networks study (Sec. 6.2), an investigative experience is required. Thus, for this study, *VERTIGo* was tested and evaluated by three investigative journalists who work for different organizations: Ecuadorian National Polytechnic School[10], University of the Hemispheres[11], and Ecuadorian Public Expenditure Observatory[12].

Usability survey: VERTIGo is a desktop application, so we operate the tool in a local environment from where we met with experts separately through online video communication[13]. The system operator had to attend to any request from the experts about the tool. Firstly, we introduced in detail *VERTIGo*'s views and functionalities. Then, using the Offshore dataset, we explained the four tasks that our system manages: Query Construction, Visualization and Exploration of Results, and Query Suggestions. At this point, the experts were free to request the construction of specific queries and explore their results. Each session had an average duration of one hour. After that, they were asked to complete a Google Form survey aimed at evaluating the functionalities of the system. The questions rate the usefulness and usability of each view/functionality on a scale ranging from 1 to 5 (1: "very useless"/"very complicated to use", 5: "very useful"/"very simple to use"). We have also collected feedback on the views and the system in general.

TABLE 2: User study evaluation: average usefulness and usability scores for each view.

Views	Usefulness (up to 5)	Usability (up to 5)
Query view (Query construction/suggestion)	4.8	5.0
Graph view (Visualization of results)	5.0	5.0
Embeddings view (Exploration of results)	5.0	4.5

Table 2 shows the average results of the user study for each view. We can note that the experts appreciate the usefulness (average score of 4.9/5) and usability (average score of 4.8/5) of our system. Considering the usefulness aspect, all three experts agree with each other that the pause and resume functionality associated to the query engine process is a convenient feature that helps users to quickly explore and navigate partial results. If the retrieved results seem irrelevant, they could build another query immediately. Experts also highlight that the process of creating a query is simple and smooth. They mention this functionality as intuitive and user-friendly. Finally, they appreciate the possibility of the system to suggest new relationships, which stimulates the creation of new possible queries. The usability of the system is also highly appreciated by experts. They mention that the exploration task is enhanced by using a heatmap showing the locations of the results on the original graph. They like that different embeddings can be shown using colors (Kelp-based visualizations). However, the Embeddings view obtained the lowest average score. As we expected, listing the results increases (overloads) the cognitive effort of a user.

Overall, *VERTIGo* meets the expectations of investigative journalists. They state that our system is well-conceived and the interactions are intuitive. A suggestion was made regarding the possibility to use different color scheme for node types. The possibility to add an option to select the color of nodes can be considered as a future improvement.

7.2 Performance of *VERTIGo*

Retrieving all the occurrences of a query in a large multilayer graph is computationally demanding [22] as it involves dealing with the subgraph isomorphism problem, which belongs to the

10. https://www.epn.edu.ec/ (accessed on November 15, 2020)
11. https://www.uhemisferios.edu.ec/ (accessed on November 15, 2020)
12. https://www.gastopublico.org/ (accessed on November 15, 2020)
13. The COVID-19 pandemic was ongoing when this study was conducted

class of NP-complete problems [32]. To reduce the System Response Time (SRT) of the query processing, *VERTIGo* implements several strategies in computational and visualization terms. First of all, it integrates the *SuMGra* [31] multilayer graph query engine that speeds the backtracking algorithm by using indexing techniques. Our system also interacts with the query engine (start, pause, and resume interactions) to allow users to explore results as soon as possible and reduce the SRT. In terms of visualization performance, we use the FM^3 technique [34] to efficiently draw large graphics (the design is rendered offline). Also, our tool does not show edges, thus does not increase the rendering time.

All these techniques are intended to provide a better user experience; however, the SRT of the system increases exponentially when the graph is large and the request query is complex (a high number of nodes). We ran the two case studies presented in Sec. 6 on a 64-bit Work Station, Intel Xeon 3.00 GHz processor with 16 GB of RAM. The first one (37,878 nodes) did not present any particular problem when performing the queries described in the case study (Sec. 6.1). Interactions such as dragging, zooming in, zooming out, and selection were seamless. In contrast, in the second case study (Sec. 6.2), composed by 406,072 nodes, the SRT increased when displaying the heatmap of the results. Also, the algorithm for labeling the nodes decreases performance (the user can disable this option from the control panel). As we expected, the performance of the system is compromised when the number of nodes increases. *VERTIGo*'s responsiveness decreases when working with $\sim$500,000 nodes on the aforementioned hardware.

8 Conclusion

In this paper, we presented *VERTIGo*, a new visual platform that allows the users to query multilayer networks, visualize/explore the obtained results and suggest new query extensions based on the underlying multilayer graph and the current query embeddings. The results are analyzed at different granularity levels and the conceived visual components support the user to focus on particular interesting areas of the multilayer graph for further inspection. The proposed system also enables the interaction between the user and the query process. The user can start, pause, and resume the query engine with the possibility to navigate/explore partially collected embeddings. Two real world examples and a user study highlight the quality of *VERTIGo* to inspect and explore information modeled as multilayer networks. Visual components and interaction techniques facilitate the user to deal with tasks commonly involved in the visual query process: *Query Construction*, *Visualization and Exploration of Query Results*, and *Query Suggestion*. We plan, as a future work, to extend *VERTIGo* for dealing with dynamic graphs where the network structure (nodes and edges) change or evolve over time.

References

[1] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, "Multilayer networks," *Journal of Complex Networks*, vol. 2, no. 3, pp. 203–271, 2014.

[2] A. Zhang, *Protein Interaction Networks: Computational Analysis*. Cambridge University Press, 2009.

[3] F. Bonchi, A. Gionis, F. Gullo, and A. Ukkonen, "Distance oracles in edge-labeled graphs," in *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2014, pp. 547–558.

[4] L. Libkin, J. Reutter, and D. Vrgoč, "TriAL for RDF: Adapting Graph Query Languages for RDF Data," in *Proceedings of the International Symposium on Principles of Database Systems (PODS)*, 2013, pp. 201–212.

[5] D. Redondo, A. Sallaberry, D. Ienco, F. Zaidi, and P. Poncelet, "Layer-Centered Approach for Multigraphs Visualization," in *Proceedings of the International Conference Information Visualisation (IV)*, 2015, pp. 50–55.

[6] B. Renoust, G. Melançon, and T. Munzner, "Detangler: Visual Analytics for Multiplex Networks," *Computer Graphics Forum*, vol. 34, no. 3, pp. 321–330, 2015.

[7] R. Bourqui, D. Ienco, A. Sallaberry, and P. Poncelet, "Multilayer Graph Edge Bundling," in *Proceedings of the Pacific Visualization Symposium (PacificVis)*, 2016, pp. 184–188.

[8] F. McGee, M. Ghoniem, G. Melançon, B. Otjacques, and B. Pinaud, "The State of the Art in Multilayer Network Visualization," *Computer Graphics Forum*, vol. 38, no. 6, pp. 125–149, 2019.

[9] D. H. Chau, C. Faloutsos, H. Tong, J. I. Hong, B. Gallagher, and T. Eliassi-Rad, "Graphite: A visual query system for large graphs," in *Proceedings of the International Conference on Data Mining (ICDM)*, 2008, pp. 963–966.

[10] R. Pienta, F. Hohman, A. Tamersoy, A. Endert, S. Navathe, H. Tong, and D. H. Chau, "VISAGE: Interactive Visual Graph Querying," in *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI)*, 2016, pp. 272–279.

[11] R. Pienta, F. Hohman, A. Endert, A. Tamersoy, K. Roundy, C. Gates, S. Navathe, and D. H. Chau, "VIGOR: Interactive Visual Exploration of Graph Query Results," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 215–225, 2018.

[12] M. Paradies, M. Rudolf, and W. Lehner, "GraphVista: Interactive Exploration of Large Graphs," *Cornell University Library*, 2015.

[13] S. S. Bhowmick, B. Choi, and S. Zhou, "VOGUE: Towards A Visual Interaction-aware Graph Query Processing Framework," in *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2013.

[14] N. Jayaram, S. Goyal, and C. Li, "VIIQ: Auto-Suggestion Enabled Visual Interface for Interactive Graph Query Formulation," *Very Large Data Base*, vol. 8, no. 12, pp. 1940–1943, 2015.

[15] E. Cuenca, A. Sallaberry, D. Ienco, and P. Poncelet, "Visual Querying of Large Multilayer Graphs," in *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, 2018, pp. 32:1–32:4.

[16] F. McGee, L. Morin, M. Stefas, S. Zorzan, and M. Ghoniem, "Layer Definition and Discovery in Multilayer Network Datasets," in *Proceedings of the International Workshop on Challenges in Multilayer Network Visualization and Analysis (MNLVIS)*, 2019.

[17] R. Pienta, J. Abello, M. Kahng, and D. H. Chau, "Scalable Graph Exploration and Visualization: Sensemaking Challenges and Opportunities," in *Proceedings of the International Conference on Big Data and Smart Computing (BIGCOMP)*. IEEE, 2015, pp. 271–278.

[18] F. Schreiber and H. Schwöbbermeyer, "MAVisto: a tool for the exploration of network motifs," *Bioinformatics*, vol. 21, no. 17, pp. 3572–3574, 2005.

[19] W. Huang, C. Murray, X. Shen, L. Song, Y. X. Wu, and L. Zheng, "Visualisation and Analysis of Network Motifs," in *Proceedings of the International Conference Information Visualisation (IV)*. IEEE, 2005, pp. 697–702.

[20] M. Pohl and A. Kerren, "Human Factors and Multilayer Networks," in *Proceedings of the International Workshop on Challenges in Multilayer Network Visualization and Analysis (MNLVIS)*, 2019.

[21] Z. Shen, K.-L. Ma, and T. Eliassi-Rad, "Visual Analysis of Large Heterogeneous Social Networks by Semantic and Structural Abstraction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 6, pp. 1427–1439, 2006.

[22] W. Han, J. Lee, and J. Lee, "Turbo$_{\text{iso}}$: Towards ultrafast and robust subgraph isomorphism search in large graph databases," in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2013, pp. 337–348.

[23] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "An improved algorithm for matching large graphs," in *Proceedings of the International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR)*. Springer, 2001, pp. 149–159.

[24] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, "Fast best-effort pattern matching in large attributed graphs," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2007, pp. 737–746.

[25] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proceedings of the International Conference on Data Mining (ICDM)*, 2002, pp. 721–724.

[26] V. Carletti, P. Foggia, A. Saggese, and M. Vento, "Introducing VF3: A New Algorithm for Subgraph Isomorphism," in *Proceedings of the*

International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR). Springer, 2017, pp. 128–139.
[27] M. Paradies, W. Lehner, and C. Bornhövd, "GRAPHITE: An extensible graph traversal framework for relational database management systems," in *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, 2015, p. 29.
[28] D. Mottin, F. Bonchi, and F. Gullo, "Graph query reformulation with diversity," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2015, pp. 825–834.
[29] P. Yi, B. Choi, S. S. Bhowmick, and J. Xu, "AutoG: A Visual Query Autocompletion Framework for Graph Databases," *Very Large Data Base*, vol. 9, no. 13, pp. 1505–1508, 2016.
[30] C. Ware, *Information Visualization: Perception for Design*. Elsevier, 2012.
[31] V. Ingalalli, D. Ienco, and P. Poncelet, "SuMGra: Querying Multigraphs via Efficient Indexing," in *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*. Springer, 2016, pp. 1–15.
[32] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Symposium on Theory of Computing (STOC)*, 1971, pp. 151–158.
[33] M. Kaufmann and D. Wagner, *Drawing graphs: Methods and models*. Springer, 2003.
[34] S. Hachul and M. Jünger, "Drawing large graphs with a potential-field-based multilevel algorithm," in *Proceedings of the International Symposium in Graph Drawing (GD)*. Springer, 2004, pp. 285–295.
[35] A. Perrot, R. Bourqui, N. Hanusse, F. Lalanne, and D. Auber, "Large Interactive Visualization of Density Functions on Big Data Infrastructure," in *Proceedings of the International Conference on Large Data Analysis and Visualization (LDAV)*, 2015, pp. 99–106.
[36] N. Smith and S. van der Walt, "A Better Default Colormap for Matplotlib," in *Proceedings of the Conference on Python in Science (SciPy)*, 2015, pp. 6–12.
[37] Y. Liu and J. Heer, "Somewhere Over the Rainbow: An Empirical Assessment of Quantitative Colormaps," in *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI)*, 2018, p. 598.
[38] C. Collins, G. Penn, and S. Carpendale, "Bubble Sets: Revealing Set Relations with Isocontours over Existing Visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1009–1016, 2009.
[39] B. Alper, N. Riche, G. Ramos, and M. Czerwinski, "Design Study of LineSets, a Novel Set Visualization Technique," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2259–2267, 2011.
[40] K. Dinkla, M. J. van Kreveld, B. Speckmann, and M. A. Westenberg, "Kelp Diagrams: Point Set Membership Visualization," *Computer Graphics Forum*, vol. 31, no. 3, pp. 875–884, 2012.
[41] E. R. Gansner and Y. Hu, "Efficient Node Overlap Removal Using a Proximity Stress Model," in *Proceedings of the International Symposium in Graph Drawing (GD)*. Springer, 2008, pp. 206–217.
[42] F. Chen, L. Piccinini, P. Poncelet, and A. Sallaberry, "Node Overlap Removal Algorithms: an Extended Comparative Study," *Journal of Graph Algorithms Applications*, vol. 24, no. 4, pp. 683–706, 2020.
[43] W. Fitzgibbon, M. Patruci, and M. Garcia Rey, "How Family that Runs Azerbaijan Built an Empire of Hidden Wealth," *International Consortium of Investigative Journalists*, 2016. [Online]. Available: https://www.icij.org/investigations/panama-papers/20160404-azerbaijan-hidden-wealth/

Erick Cuenca is an assistant professor at Yachay Tech University, Urcuquí, Ecuador. He is also a member of the data mining and visualization research group (ADVANSE) of the Laboratory of Computer Science, Robotics and Microelectronics of Montpellier (LIRMM). His research focuses on datastream and multivariate graph visualization. Cuenca received a PhD in computer science from the University of Montpellier, France.

Arnaud Sallaberry is an assistant professor at Paul Valéry University of Montpellier, France. He is the head of the data science research group (AMIS). He is also a member of the data mining and visualization research group (ADVANSE) of the Laboratory of Computer Science, Robotics and Microelectronics of Montpellier (LIRMM). His research interests include information visualization, visual analytics, and graph drawing. Sallaberry received a PhD in computer science from the University of Bordeaux, France.

Dino Ienco is a Research Scientist at the INRAE Institute, France and he is an associated member of the data mining and visualization research group (ADVANSE) at the Laboratory of Computer Science, Robotics and Microelectronics of Montpellier (LIRMM). His research interests include data science, data mining and machine learning techniques with applications on spatio-temporal information and graph data. Dino Ienco received a PhD in Computer Science from the University of Torino, Italy.

Pascal Poncelet is a full professor at the University of Montpellier, France, and head of the data mining and visualization research group at the Laboratory of Computer Science, Robotics and Microelectronics of Montpellier (LIRMM). His research interests include advanced data analysis techniques for emerging applications, data mining techniques, and new algorithms for mining patterns. Poncelet received a PhD in computer science from the University of Nice-Sophia Antipolis.