Interactive graph query language for multidimensional data in Collaboration Spotting visual analytics framework

Adam Agocs*
CERN, CH-1211 Geneva 23, Geneva, Switzerland
Jean-Marie Le Goff‡
CERN, CH-1211 Geneva 23, Geneva, Switzerland
Dimitrios Dardanis†
CERN, CH-1211 Geneva 23, Geneva, Switzerland
Dimitrios Proios‡
CERN, CH-1211 Geneva 23, Geneva, Switzerland

ABSTRACT
Human reasoning in visual analytics of data networks relies mainly on the quality of visual perception and the capability of interactively exploring the data from different facets. Visual quality strongly depends on networks’ size and dimensional complexity while network exploration capability on the intuitiveness and expressiveness of user frontends.

The approach taken in this paper aims at addressing the above by decomposing data networks into multiple networks of smaller dimensions and building an interactive graph query language that supports full navigation across the sub-networks. Within sub-networks of reduced dimensionality, structural abstraction and semantic techniques can then be used to enhance visual perception further.

Index Terms: Information Systems [H.2.3]: Language—Query language Image Processing and Computer Vision [I.4.10]: Image Representation—Multidimensional

1 INTRODUCTION
According to an English idiom, "A picture is worth a thousand words". Visual analytics aims to combine the power of visual perception with high performance computing in order to support human analytical reasoning. Since Pak Chung Wong and Jim Thomas published their article named "Visual Analytics" in 2004, visual analytics has been widely used in various fields such as biology or national security but also in other fields such as climate monitoring or social networks analysis, the field originally addressed by the Collaboration Spotting project. Networks built out of interconnected elements contained in datasets and represented as multidimensional, directed and labelled graphs are a natural means of representing data for visual analytics.

Graphs as database models and graph query languages defined over these models have been investigated for some 30 years (Wood, P.T. [13]). These models and languages have been used in many applications using a wide spectrum of data (e.g. biology, social network and criminal investigation data), clearly indicating that the combination of visual analytics with graph query languages has become quite popular.

According to Wong, P.C. et al [41], one of the biggest challenges in visual analytics is User-Driven Data Reduction which calls for "a flexible mechanism that users can easily control according to their data collection practices and analytical needs". This essentially entails an improvement of the visualization clarity and an escalation of data processing performances irrespective of the increasing complexity of the data over the years. To meet this challenge, semantic and structural abstraction techniques such as clustering, collapsing and extraction and demonstration of relationships among graph entities can be used at the expense of a loss of information on the network content [36].

The approach taken in the Collaboration Spotting project is to reduce the dimensional complexity of data networks while maintaining the information about their content. It consists in decomposing multi-dimensional, directed and labelled graphs into multiple directed and weighted graphs of lesser dimensions - named views - and in building an interactive graph query language that supports user-specified views and full navigation across the data networks using these views as a support to the operations of the language. Within a view, structural abstraction techniques can then be used to enhance the visual perception further. This approach is quite similar to the concept of blueprint where the architectural plan is distributed across different views. The main advantage of this approach is that it combines Visual graph representation and User interactions at the graph query language level.

Section 2 gives a short overview on visualisation techniques for visual analytics (focusing on social networks) and on graph query languages fit to data networks. Section 3 gives a short description of the mathematical background supporting the approach. Section 4 introduces how views are constructed and Section 5 shows how the basic operations of the query language enable users to conduct their analysis. In Section 6 the use-case that inspired the Collaboration Spotting project and the graphical query language are presented. This paper ends with conclusions and future work in Section 7.

2 RELATED WORK
The related work is twofold since it combines multiple visual analytics techniques with the power of graph query languages. In the last 15 years, a lot of visual analytics articles were published with the aim of showing processes of transformation of multidimensional data into node-link diagrams [24][38].

A lot of articles have been published, especially on the coordinated multiple views topic, which introduces a visual analytics paradigm supported by an interactive query language or by a set of operations. These articles can be divided into four different groups:

- OLAP [15] inspired paradigms that are using operations like slice, roll-up, dice, etc. The most relevant papers are PivotGraph [39], ScatterDice [10] (and GraphDice [4]), Matrix-Cube [1] and Orion [19].

- Relational algebra-related solutions such as Cross-filter views [40] which uses grouping, filtering, projection and selection operations, Polaris [37] that introduces and maps its algebra to SQL and Ploceus [28] which works with first-order logic language.

- Other solutions such as Cross-filter views with hypergraph query language [34], JUNG [31] and Gephi [3] that allow users to use other programming languages (JAVA in these cases).
3 Basic Graph and Views

Let graph \( G \) be a directed, labeled graph defined as a four-element tuple \( G = (V,E,L,\alpha) \) where \( V \) represents a set of vertices and \( E \subseteq V \times V \), a set of edges defined as a subset of the Cartesian products of these vertices. \( L \) is a set of vertex labels and \( \alpha : V \rightarrow L \) is a mapping function from vertices to the corresponding labels. Figure 1 shows an example of such a graph. We define the reachability graph over graph \( G \) as \( G_{\text{reachability}} = (L,E_{\text{reachability}}) \) where vertices are labels of graph \( G \), \( E_{\text{reachability}} \subseteq L \times L \) is defined as the Cartesian product of the labels where any two vertices of \( G_{\text{reachability}} \) are connected if and only if there exists two connected vertices in graph \( G \) and their respective labels correspond to the two vertices of graph \( G_{\text{reachability}} \). Graph \( G_{\text{reachability}} \) is a description of graph \( G \), it is also called the graph schema of graph \( G \). Graph schema helps users view graph \( G \) via different sub-graphs of lesser dimensionality using labels of \( G \) as dimensions and facilitates the generation of approximately optimal user-defined graph queries. Let graph \( G_{\text{pattern}} = (V_{\text{pattern}}, E_{\text{pattern}}) \) be a graph pattern where \( V_{\text{pattern}} \subseteq V \) and \( E_{\text{pattern}} \subseteq E_{\text{reachability}} \cap V_{\text{pattern}} \times V_{\text{pattern}} \). To process the answer to a graph query, one needs to find all possible isomorphic subgraphs of \( G \) that are homomorphic to a graph pattern \( G_{\text{pattern}} \) corresponding to the query. This is a graph pattern matching problem, a well-known part of graph theory \([13]\). In this case, one defines graph \( G' = (V', E', L', \alpha') \), a subgraph of graph \( G \) as a sample matching the graph pattern \( G_{\text{pattern}} \) if and only if:

\[
\forall v' \in V' \exists v \in V_{\text{pattern}}. \alpha(v') = v.
\]

\[
\forall (u', v') \in E'. (\alpha(u'), \alpha(v')) \in E_{\text{pattern}}.
\]

The answer to a graph query is a view containing the set of subgraphs of \( G \) matching \( G_{\text{pattern}} \). To build such a view, one needs first to introduce the graph pairing function \( \text{pair} \) and the set \( \text{Pattern} \). Let \( G_{\text{pattern}}_1 \) and \( G_{\text{pattern}}_2 \) be two graph patterns. These graph patterns are paired if:

- \( V_{\text{pattern}}_1 = V_{\text{pattern}}_2 \) and
- \( \exists a,b \in V_{\text{pattern}}_1 : \text{path}(a,b) \in E_{\text{pattern}}_1 \) and \( \text{path}(a,b) \notin E_{\text{pattern}}_2 \), \( \text{path}(b,a) \notin E_{\text{pattern}}_1 \) and \( \text{path}(a,b) \notin E_{\text{pattern}}_2 \setminus \text{path}(b,a) \).

Where a path is an alternate non-empty sequence of vertices and edges, starting and ending with vertices and requiring that all edges and vertices be distinct from one another. \( \text{path}(a,b) \in E_{\text{pattern}} \) indicates that all edges of this path are in set \( E_{\text{pattern}} \). The \( \text{pair} \) function is defined as

\[
\text{pair}(G_{\text{pattern}}) := \begin{cases} G_{\text{pattern}} \quad & \text{if } G_{\text{pattern}} \text{ is a pair of } G_{\text{pattern}} \\ (\emptyset, \emptyset) & \text{else.} \end{cases}
\]

And \( \text{Pattern} \), the set of these pairs is defined as \( \text{Pattern} := \{ (g, g') | g, g' \text{ are patterns}, g' = \text{pair}(g) \} \).

A view of graph \( G \) is defined as a six-element tuple \( G_q = (C_q, B_q, E_q, L_q, \epsilon_q, \nu_q) \) where:

- \( C_q \subseteq V \), \( L_q := \{ \alpha(v) | v \in C_q \} \),
- \( B_q \subseteq V \), \( L_q := \{ \alpha(b) | b \in B_q \} \),
- \( L_q \subseteq L \) and \( L_q = L' \cap L_q \),
- \( E_q := \{(u,v) | u,v \in C_q, (u,v) \in G_q \} \subseteq G \), \( \exists (G_{\text{pattern}}, \text{pair}(G_{\text{pattern}})) \in \text{Patterns} : G' \text{ matches to } G_{\text{pattern}}, G'' \text{ matches to } \text{pair}(G_{\text{pattern}}) \), \( \exists b \in B_q : \text{path}(u,b) \in G', \text{path}(b,v) \in G'' \},
- \( E_q := \{ (u,v) | u,v \in C_q, \exists G', G'' \subseteq G \), \( \exists (G_{\text{pattern}}, \text{pair}(G_{\text{pattern}})) \in \text{Patterns} : G' \text{ matches to } G_{\text{pattern}}, G'' \text{ matches to } \text{pair}(G_{\text{pattern}}) \), \( \text{path}(u,b) \in G', \text{path}(b,v) \in G'' \}) \),
- \( \nu_q := C_q \Rightarrow \rho(B_q) \), \( \nu_q(u) = \{ b | b \in B_q, \exists G' \subseteq G \), \( \exists (G_{\text{pattern}}, \text{pair}(G_{\text{pattern}})) \in \text{Patterns} : G' \text{ matches to } G_{\text{pattern}}, \text{path}(u,b) \in G'' \}, \)

The use of multiple graph patterns for the construction of graph \( G_q \) is required since the cardinality of set \( L_q \) and set \( L_q \) is not necessary equal to 1 (see details in Section \( 2 \).\( 1 \)). To ease the reading, graph \( G_q \) is noted \( G_q^{L_q} \) to refer directly to the set of labels used in the construction of the view. Also, in practice, we use an aggregation function on edges, respectively on vertices in graph \( G_q \) for determining their respective weights instead of the elements in set \( B_q \) (for instance, the number of elements). Figure 2 shows an example of a view.

4 Graph Creation from User Interactions

In this section, we introduce how the graph patterns and views can be created as a result of the following user interactions:

- Selection of different nodes in the current view,
- Removal of all vertices with the same label selected in one of the previous views,
- Navigation from one view to another.

Users can modify set \( L_q \) and set \( L_q \) when performing any of the above interactions. Let \( F \subseteq V \) be the set of vertices corresponding to a user selection, we define from \( F \):

\[
L_F := \{ l \in L | \exists f \in F : \alpha(f) = l \}
\]

which contains the labels of nodes in set \( F \) and,
1. This section shows how to construct a graph pattern with set \( L \). The two graph patterns are \( G_{\text{pattern}} = (\{\text{label1}, \text{label3}\}, \{(\text{label1}, \text{label3})\}) \) and \( \text{pattern} = (\{\text{label3}, \text{label1}\}, \{(\text{label3}, \text{label1})\}) \).

2. \( F_{L'} := \{ f \in F | \alpha(f) \in L' \} \) with \( L' \subseteq L \), a subset of set \( F \), restricted to vertices having their respective labels in set \( L' \).

In order for set \( F \) to operate as a filter, the matched sample definition of Section 3 has to be restricted by requiring that \( \forall v' \in V', \alpha(v') \in L_{F} \Rightarrow v' \in F \). Example 1 below shows the content of \( L_{F} \) for user selection \( F = \{ v_4, v_6, v_7, v_{13} \} \) from the graph \( G \) depicted in Figure 1.

**Example 1**

\[
\begin{align*}
F &= \{ v_4, v_6, v_7, v_{13} \} \\
L_{F} &= \{ \text{label2}, \text{label3}, \text{label5} \}
\end{align*}
\]

4.1 Graph pattern construction

This section shows how to construct a graph pattern with set \( L_{F} \) containing all the labels of vertices in set \( F \). We exploit the fact that graph patterns are actually only needed when constructing edges in \( G_{L_{F}}^{\Delta} \) and their respective weights. A pair of graph patterns are required for each combination of labels in set \( L_{C} \) and set \( L_{B} \) since the path direction between vertices from set \( L_{C} \) and set \( L_{B} \) are different due to the construction of edges between vertices of \( C_{q} \) and vertices of \( B_{q} \). Each pattern has to satisfy the following criteria:

- It must be a connected and directed graph,
- It must be minimal,
- Labels from set \( L \setminus L_{F} \) can be used as intermediate vertices in the pattern.

These requirements exactly fit a Steiner Minimal Tree problem [23], known to be NP-complete [14] and for which we use a minimal spanning tree solver as an approximation algorithm. Algorithm 1 describes the full process of pattern generation. Figure 2 shows the graph schema of graph \( G \) depicted in Figure 1 and the generated patterns.

4.2 Connecting user interactions and views

Now that graph patterns (Patterns) have been created using set \( F \), set \( L_{C} \) and set \( L_{B} \), one can introduce the \textit{gen} function \( \text{gen} : \mathcal{P}(V) \times \mathcal{P}(L) \times \mathcal{P}(L) \rightarrow G_{L_{F}}^{\Delta} \) that generates views from user interactions.

**Algorithm 1 Pattern generator algorithm**

1. \( \text{function PATTERN}\_\text{GENERATOR}(F_{L}, L_{B}, L_{C}) \)
2. \( \text{Patterns} \leftarrow \emptyset \)
3. \( B \leftarrow L_{B} \)
4. \( \text{while } B \neq \emptyset \) do
5. \( \text{from, } B \leftarrow B \setminus \{ \text{from} \} \)
6. \( E \leftarrow L_{C} \)
7. \( \text{while } E \neq \emptyset \) do
8. \( \text{to, } E \leftarrow E \setminus \{ \text{to} \} \)
9. \( \text{Left} \leftarrow \text{SpanningTree}(F_{L} \cup \{ \text{from, to} \}, \text{from, to}) \)
10. \( \text{Right} \leftarrow \text{SpanningTree}(F_{L} \cup \{ \text{from, to} \}, \text{to, from}) \)
11. \( \text{Patterns} \leftarrow \text{Patterns} \cup \{ \text{Left, Right} \} \)
12. \( \text{end while} \)
13. \( \text{end while} \)
14. \( \text{return Patterns} \)
15. \( \text{end function} \)

Figure 2: Example of a view where \( C_{q} = \{ v_6, \ldots, v_9 \}, B_{q} = \{ v_1, \ldots, v_3 \} \). The two graph patterns are \( G_{\text{pattern}} = (\{\text{label1}, \text{label3}\}, \{(\text{label1}, \text{label3})\}) \) and \( \text{pattern} = (\{\text{label3}, \text{label1}\}, \{(\text{label3}, \text{label1})\}) \).

Figure 3: On the left hand-side, the graph schema of graph \( G \) in the example of Figure 1. On the middle and on the right hand-side, an example of a graph pattern pair for set \( F \) (Example 1) with \( L_{C} = \{ \text{label4} \} \) and \( L_{B} = \{ \text{label1} \} \).

\( (F \subseteq \mathcal{P}(V) \text{ and } L_{C}, L_{B} \subseteq \mathcal{P}(L)) \) as \( \text{gen}(F, L_{C}, L_{B}) := F G_{L_{F}}^{\Delta} = (F C_{q}, F B_{q}, E_{q}, v_{q}, e_{q}) \) where

\[
F C_{q} := \begin{cases} V \cap F_{L_{C}} & \text{if } V \cap F_{L_{C}} \neq \emptyset \\ V_{L_{C}} & \text{else} \end{cases}
\]

are the vertices of graph \( F G_{L_{F}}^{\Delta} \) and

\[
F B_{q} := \begin{cases} b \in V' | L_{B} & \exists G' = (V', E', L, \alpha) \subseteq G: \\ G' \text{ matches to } G_{\text{pattern}}, \forall v' \in V': \\ \{v' \in F \text{ or } \alpha(v') \notin L_{F}\} \end{cases}
\]

are the “interconnection” vertices: The other members of the six-tuple \( G_{q} \) are unchanged since

- labels (set \( L_{q} \)) are not modified and since
- edge definition (set \( E_{q} \)) and weighting functions (\( v_{q} \) and \( e_{q} \)) only depend on set \( F C_{q} \) and set \( F B_{q} \).

5 OPERATIONS ON GRAPHS

User interactions will result in the following graph operations:

- Selection: The user selects nodes on the view,
• **Expansion**: The user expands a view by removing in his previous selection, vertices having the same labels.

• **Navigation**: The user navigates from a view to another.

To define these operations one needs first to introduce the concepts of visual equivalence and minimal views since there can be views with vertices of null weight that are hidden to the user and hence non-selectable. Let $F_1$ and $F_2$ be two different filters on the same view complying with $F_1 \setminus F_1|^{L_C} = F_2 \setminus F_2|^{L_C}$. In essence, this means that there is no difference in the sets of vertices with labels contained in $L \setminus L_C$ which technically should be empty. View $F_1 \circ G|^{L_C}$ and view $F_2 \circ G|^{L_C}$ generated using $F_1$ and $F_2$ are said to be visual equivalent if and only if

**Definition 2 (Vis-equivalent)**

$$F_1 \circ G|^{L_C} \sim F_2 \circ G|^{L_C} \iff \forall v \in V_1 \setminus V_2 : \nu_1(v) = \emptyset, \forall v' \in V_2 \setminus V_1 : \nu_2(v') = \emptyset,$$

where $V_1$ ($V_2$) represents the vertices of view $F_1 \circ G|^{L_C}$ ($F_2 \circ G|^{L_C}$). Intuitively visual equivalence guaranties that vertices that are not common to two views have empty weights. It provides equivalence classification on views. It is easy to prove that for each class of views there is only one which does not have vertices with empty weights. This view is called the minimal view.

### 5.1 Selection on graphs

Let $F_{select}$ be the set of user selected nodes within a view, $F_{select} \subseteq V$ and $F_{select} \subseteq V'$ where $V'$ is a set of vertices from the minimal view which is visual-equivalent to graph $F \circ G|^{L_C}$. The selection operator $\sigma : G_0 \times \mathcal{P}(V) \rightarrow G_0$ is defined as

**Definition 3 (Selection)**

$$\sigma(F \circ G|^{L_C}, F_{select}) := \begin{cases} \gen(F \cup F_{select}, L_C, L_B) & \text{if } F \cap F_{select} = \emptyset \\ \gen(F \setminus F_{select}, L_C, L_B) & \text{else} \end{cases},$$

where $F_{select} = \{ f | f \in F, \alpha(f) \in L_C \}$. It is to be noted that at view creation the selection operator has been used with a more general definition of the $\gen$ function.

### 5.2 Expansion on graphs

The expansion operator $\xi$ is in some sense the “invert” or the selection operator. It is defined as

**Definition 4 (Expansion)**

$$\xi(F \circ G|^{L_C}, L_C) := \gen(F \setminus L_C, L_C, L_B).$$

The expansion operator changes view when $L_C \neq L_C$ and remove all vertices in set $F$ that are labelled with labels in $L_C$.

### 5.3 Navigation through graphs

By selecting a subset of labels from $L_C$ one can build views of graph $G$ with reduced dimensional complexity. Navigation across views is required to enable users to apprehend the full graph $G$. Therefore the navigation function $\eta$ goes from view $F \circ G|^{L_C}$ to a view labelled as $L_C'$ and $L_B'$ and is defined as:

**Definition 5 (Navigation)**

$$\eta(F \circ G|^{L_C}, L_C', L_B') := \gen(F, L_C', L_B') \quad \text{for} \quad \eta(F \circ G|^{L_C}) = \emptyset.$$
6.2 Storing data in a graph database (Neo4j)

Graph $G$ is stored in a Neo4j graph database [30], in which individual metadata records are stored as subgraphs of labelled vertices using Published item, Organisation, Journal Category, Author Keyword, City, Region and Country as labels. Figure 4 represents the reachability graph (graph schema) of this network. Besides these labels, additional labels have been introduced to support user authentication and authorisation (User) and searches (Graph and Technology). Searches use full text indices of the Apache Lucene project [29] that have been integrated into the Neo4j database as legacy indices [30].

![Figure 4: The database schema (reachability graph)](image)

Light color nodes represent nodes uploaded by the data administrator and the dark nodes are created by the system itself by using search and authentication modules.

6.2.1 Statistic of our graph data

Searches on publications and patents metadata records from the 2000 - 2014 period can be performed. The resulting data network contains 45 million vertices and 150 million edges. Its breakdown is given in Table 1 and Table 2.

Table 1: Number of nodes by node labels

| Type of nodes | Number of nodes |
|---------------|-----------------|
| Patents       | 15,000,442      |
| Publications  | 20,087,904      |
| Organisations | 2,918,060       |
| Author Keywords | 8,193,604   |
| Subject Categories | 230    |
| Cities        | 7,741           |
| Regions       | 946             |
| Countries     | 128             |
| $\Sigma$      | 46,209,055      |

As can be noticed the number of region edges is smaller than the number of country edges due to the use of the 2nd level of Nomenclature of Territorial Units For Statistic [11] created by the European Commission.

6.3 Navigation

The entry point for this use case is individual users. Using the terminology introduced above, the initial user interaction set $F$ contains user IDs.

7 CONCLUSION AND FUTURE WORK

The current version of Collaboration Spotting running at CERN [8] addresses the implementation of the concepts using patents and publications metadata records. It is a new experimental service that aims to provide the High Energy Physics community (such as HEPTech [20]) with information on Academia & Industry main players active around key technologies, with a view to fostering more inter-disciplinary and inter-sectoral R&D collaborations, and giving the procurement service the opportunity of reaching a wider selection of high-tech companies for bidding purposes. Collaboration Spotting is generic in its concepts and implementation. It can support visual analytics of any kind of data and its backend is implemented using Neo4j graph database [30]. Conference papers, technical & business news, trademarks & designs and financial data are amongst the data targeted to enrich the information on technologies that one can obtain from publications and patents. The choice of data sources will depend on users’ priorities. The tool can be of use to other communities, in particular in dentistry [27] but also to policy makers and investors if data in the labelled graph is enriched with technical & business news and financial data. Collaboration Spotting also addresses other types of data such as compatibility and dependency relationships in software and meta-data [5][55] of the LHCb experiment at CERN.

As an interactive graph query language, Collaboration Spotting is intended to provide a fully customisable visual analytics environment. In the current version data processing supports searches and contextual queries. In the future, labelled & directed relationships and attributes on nodes will be included in the labelled property graph representation of the data network and the processing will be extended to more complex operations directly on the graph resulting from searches and queries with a view to enhancing the visual perception of users.

ACKNOWLEDGMENTS

REFERENCES

[1] B. Bach, E. Pietriga, and J.-D. Fekete. Visualizing dynamic networks with matrix cubes. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI ’14, pp. 877–886. ACM, New York, NY, USA, 2014. doi: 10.1145/2556288.2557010

[2] P. Barceló Baeza. Querying graph databases. In Proceedings of the 32Nd Symposium on Principles of Database Systems, PODS ’13, pp.
(a) Technology view ($L_C = \{\text{Technology}\}, L_B = \{\text{Published Item}\}$) of a user selecting two technologies ($F = \{\text{Visual Analytics, Language}\}$) and navigating to $L_C = \{\text{Subject Category}\}$ view.

(b) Subject Category view ($L_C = \{\text{Subject Category}\}$) for the selected technologies.

(c) Selecting a cluster in the Subject Category view ($F = \{\text{Visual Analytics, Language, Linguistics... Rehabilitation}\}$) and expanding the view to go back to the Technology view ($L_C = \{\text{Technology}\}$).

(d) Technology view with $F = \{\text{Language, Linguistics... Rehabilitation}\}$ filter.

Figure 5: Example of operations; navigation, selection and expansion on views.
