IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 August 2023, accepted 28 August 2023, date of publication 20 September 2023, date of current version 27 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3317369

RESEARCH ARTICLE

Indexing Structures for the Efficient Multi-Resolution Visualization of Big Graphs

MARCO MESITI, MARIO PENNACCHIONI, AND PAOLO PERLASCA
Department of Computer Science, Università di Milano, Milan, Italy

Corresponding author: Marco Mesiti (marco.mesiti@unimi.it)

This work was supported by the National Center for Gene Therapy and Drugs Based on RNA Technology through the NextGenerationEU Program under Grant G43C22001320007.

ABSTRACT Nowadays there is a great interest in the visualization of property graphs to make their navigation, inspection, and visual analysis easier. However, property graphs can be quite large and their rendering on web browsers can lead to a dark cloud of points that is difficult to visually explore. With the aim of reducing the size of the visualized graph, several approaches have been proposed for substituting clusters of related vertices with aggregated meta-nodes and introducing meta-edges among them, but they usually consider the graph in main-memory and do not adopt efficient data structures for extracting parts of it from the disk. The purpose of this paper is to optimize the preparation of the graph to be visualized according to a certain resolution level by introducing refined data structures and specifically tailored algorithms. By means of them, the rendering time is reduced when changing the current visualization through zoom-in, zoom-out, and related operations. Starting from a cluster hierarchy that represents the possible aggregations of graph nodes, in the paper we characterize a visualization according to a horizontal slice of the hierarchy and propose indexing structures and incremental algorithms for quickly passing to a new visualization with minimal changes of the current one. In this process, we ensure a consistent and efficient aggregation of addictive properties associated with nodes and edges. An extensive experimental analysis has been conducted to assess the quality of the proposed solution.

INDEX TERMS Property graphs, node indices, edge indices, aggregations according to a cluster hierarchy, multi-resolution visualization, zoom-in and zoom-out operations, incremental algorithms.

I. INTRODUCTION

Property graphs are nowadays largely used in different contexts like the bio-molecular domain (for the representation of proteins interaction), smart cities (for urban traffic management), social networks (for maintaining individual relationships), and semantic web (for knowledge representation). These graphs do not simply report the network topology but also properties of the individuals and their relationships. Properties can be minimal information (e.g. weights) but also records of key-values pairs or complex vectors. The visual exploration of such graphs [1], [2] is of paramount importance for detecting and analysing useful hidden patterns. However, when property graphs tend to be large, a dark cloud of points is drawn making impossible to discern its content. Moreover, visualization libraries turn out to be unacceptable slows (especially when used in web applications) and the user experience is compromised.

Among the different approaches proposed for dealing with graphs of big size (e.g. [3], [4], [5] for biomolecular networks visualization, [1], [6], [7], [8] for the proper visualization and interactive analysis of complex graphs), the visualization based on the identification of clusters/communities/motifs of nodes that are highly related (or tightly connected) is gaining momentum and exploited in different contexts ([9], [10], [11]). We have also recently proposed an approach for the exploration of big bio-molecular networks that relies on the construction of a hierarchy of communities on the graph and allows the visualization of the graphs at different levels of resolution by exploding the identified communities [12]. In this kind of visualization, *meta-nodes* are introduced in the visualization for representing clusters of nodes and *meta-edges* for representing relationships among

The associate editor coordinating the review of this manuscript and approving it for publication was Rahim Rahmani.

atomic-nodes and meta-nodes. Meta-nodes can be exploded (zoom in operation) or collapsed (zoom out operation) for changing the current visualization and provide a better rendering of the network that the user wishes to navigate and explore. For example, starting from the network reported in Figure 1, a cluster hierarchy can be identified (as the one in the right-hand side of the figure) and used for the graph visualization at different levels of resolution starting from meta-nodes in the higher levels of the hierarchy (reported in the top part of Figure 2) to visualizations containing also atomic-nodes belonging to the original graph (bottom part of Figure 2). In this way, the user can decide the part of the network to be expanded or collapsed and thus pointing out structural properties of the entire graph.

Even if many research works have been focused on identifying the right visualization artifacts and interaction strategies for the proper visualization of the meta-nodes and their relationships, the issue of identifying data structures and algorithms for handling big graphs that cannot be maintained in main-memory still needs to be properly addressed. This is definitely relevant for reducing the rendering time when moving to a new resolution and when networks are stored in secondary memory (independently from the model adopted for the representation). Moreover, even if many approaches consider the presence of a cluster hierarchy, a characterization of the current visualization in its terms is missing. This is of great relevance in order to identify incremental algorithms for the application of the zoom in/out operations. Finally, there is the need to maintain consistent and efficient the aggregation of addictive properties in the passage from a resolution to another.

In this paper, we focus on the extraction of the graph to be visualized from a database in secondary memory, and on the generation of visualizations according to a hierarchy of clusters with which the multi-resolution partitioning of the graph can be defined. Our work relies on the following observations. The first observation is that when providing a graph visualization at a certain level of resolution, we are considering meta-nodes that form a chain in the tree hierarchy (as highlighted in Figure 2). The interesting chains are those that horizontally cut the hierarchy into two parts. Each one of these chains, named *slices*, corresponds to a *resolution level* of visualization, and by using this concept it becomes natural the passage from one resolution level to another and incremental algorithms can be easily devised. The second observation is that the containment relationships among clusters can be easily identified by exploiting numeric intervals associated with each node of the cluster hierarchy and by associating to the graph node the pair of numbers corresponding to the cluster leaf it belongs to. Relying on the proposed indexing structures, the main $\texttt{zoom}_{\texttt{IN}}$ and $\texttt{zoom}_{\texttt{OUT}}$ operations are computed more quickly. A final observation is that additive attributes associated with nodes and edges can be pre-aggregated in our indices, often avoiding the access to the original graph in the generation of a visualization.

Starting from these observations, in this paper we define the concept of *slice* along with its properties that can be exploited in the application of several visualization operations (i.e. the $\texttt{zoom}_{\texttt{IN}}$ and $\texttt{zoom}_{\texttt{OUT}}$ operations and also other ones introduced in this paper). All these operations are made efficient by the introduction of our indexing structures. These structures enhance those introduced in [12] by considering addictive properties and a different numeric scheme. Moreover, a detailed analysis of their properties and time complexity are reported with their positive effects on the execution of navigation operations. An extended evaluation of the performances of the proposed approach has been also conducted with real graphs showing that the approach can be exploited in web applications dealing with graphs with millions of edges and produce compact visualizations through meta-nodes and meta-edges in a few seconds.

The paper is organized as follows. Section II presents our data structures. Section III presents the concept of slice and its use in the generation of the graph at a given resolution. Section IV deals with our navigation operations and their time complexity. Section V reports the experimental results while a comparison with related work is discussed in Section VI. Section VII draws our conclusions and outlines future research directions. In the supplementary, proofs of the lemmas and theorems introduced in the paper are reported.

II. GRAPHS, CLUSTER HIERARCHY AND INDEXES

A. GRAPH

Given a set of labels $\mathcal{L}$ and values $\mathcal{V}$, a *property graph* is an unordered graph $G = (V, E, \eta, \xi)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. Properties are values associated with nodes and edges are represented by means of functions $\eta : V \times \mathcal{L} \rightarrow \mathcal{V}$ and $\xi : E \times \mathcal{L} \rightarrow \mathcal{V}$. Given a vertex $v \in V$, $v.l$ represents the value associated with the property l of the vertex v (analogously for an edge $e \in E$). We assume the existence of a special label used for uniquely identifying vertices (denoted $v.\texttt{id}$). Even if properties on edges/vertices can be of any type, we assume the presence of additive properties [13] (like `weight` on edges) that can be aggregated (with commutative and associative functions like `max`, `min`, `sum`) as described in the next section.

Example 1: Figure 1 shows an excerpt of a protein-to-protein interaction network of the *arabidopsis thaliana* organism. Each node is characterized by different properties: a unique identifier, the protein name, the corresponding organism, and the probability of expressing a GO function. For the sake of readability, in the figure we have highlighted these properties only for node 1. Edges represent their functional relationships and their strength is reported in the `w`(eight) property. In this graph, a hierarchical clustering method has been applied to cluster similar proteins and highlighted in two ways: nodes belonging to the same leaf cluster present the same color; dashed lines delimit the aggregation of clusters. □

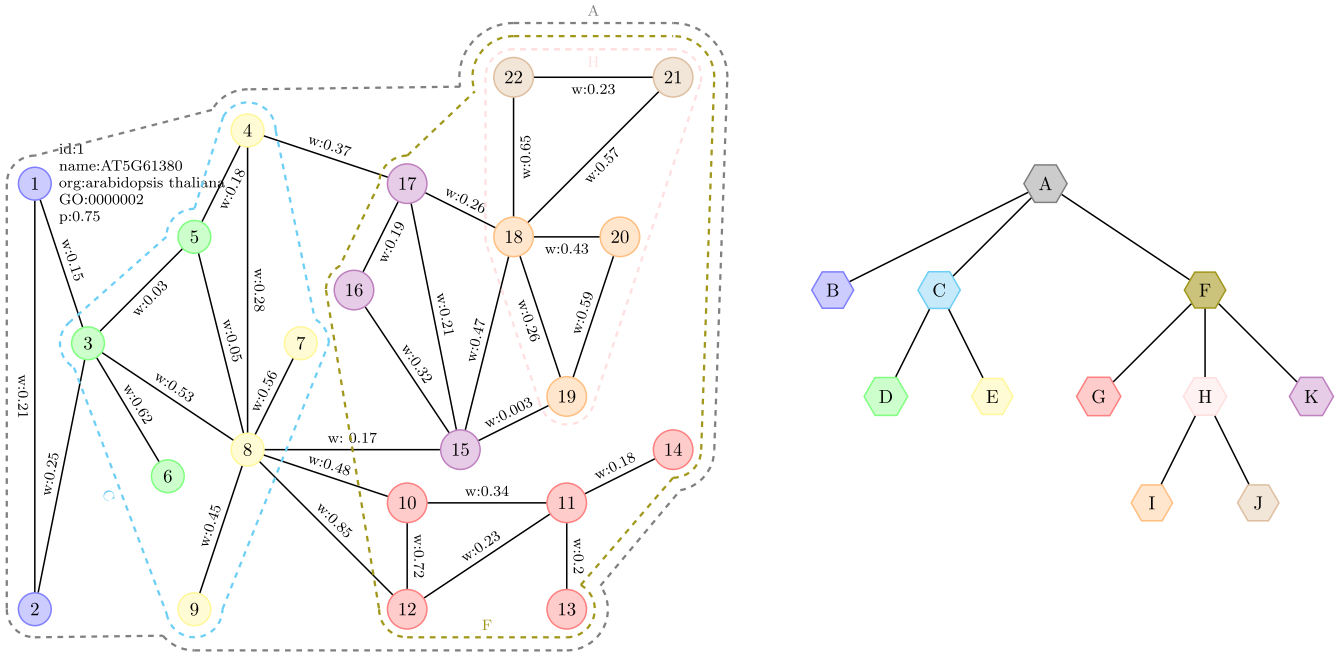


FIGURE 1. (a) A property graph on which a hierarchy cluster-based detection algorithm has been applied. Nodes with the same color have been clustered together. Moreover, dashed lines aggregate clusters at higher levels of the hierarchy. (b) The corresponding cluster hierarchy.

B. CLUSTER HIERARCHY

A cluster hierarchy can be induced on a graph G by means of different kinds of hierarchical cluster detection algorithms (see Section VI) by taking into account the node and edge properties and the network topology. A cluster hierarchy can be represented as a ranked labeled tree $C = (c_r, V_C, E_C)$, where V_C is the set of nodes representing clusters, c_r is the root of the hierarchy, and $E_C \subset V_C \times V_C$ represents the inclusion relationship (e.g. $(c, \bar{c}) \in E_C$ means that $\bar{c}$ is contained in c). A peculiarity of this tree is that each internal node presents at least two children (i.e. a cluster is split in at least two sub-clusters) that are ranked. In addition, $L(V_C)$ denotes the leaf nodes of C, $\mathcal{F} : V \rightarrow L(V_C)$ is a surjective clustering function that identifies, for each vertex in the graph G, the leaf node of tree C it belongs to. Nodes in $L(V_C)$ can be ordered relying on the structure of the cluster hierarchy. This ordering will be exploited for the creation of the indices in the following section. For each $c \in L(V_C)$, $cluster(c)$ denotes the not empty set $\mathcal{F}^{-1}(c)$. Of course $cluster(c)$ can be easily extended to each node of C using the inclusion relationship. For each internal vertex $c \in V_C$, $L_c(V_C)$ denotes the leaves of the subtree rooted in c. To facilitate the reading, we introduce the following notations on trees. For $c, c' \in V_C$:

- $child(c)$ is the list of children of c, eventually empty when c is a leaf node;
- $sibling(c)$ is the list of siblings of c;
- $path(c) = (c_r, .., c)$ is the path from the root c_r to c;
- $level(c)$ is the level of c starting from 0, i.e. $|path(c)| - 1$
- $desc(c)$ is the set of descendants of c;
- $parent(c)$ is the parent of c (undefined for c_r);
- c and c' are *disjoint* (denoted $c \not\sim c'$) when $c \notin desc(c')$ and $c' \notin desc(c)$.

The right-hand side of Figure 1 shows the cluster hierarchy that we use in our running example.

C. LEAFINDEX STRUCTURE

For making efficient determining the cluster an atomic-node belongs to and when a cluster is contained into another cluster, we introduce an indexing structure named `LeafIndex` that associates with each node c of the cluster hierarchy two kinds of information. First, a pair of natural numbers that are used for determining the inclusion relationships among clusters and the level of c in the hierarchy. Second, the number of vertices and internal edges of G that belong to the cluster, and a set of tuples $\{(l_1, n_1, s_1, mx_1, mn_1), \ldots, (l_k, n_k, s_k, mx_k, mn_k)\}$ for the addictive properties $l_1, \ldots, l_k$ that can be specified for the vertices of G. A tuple $(l_i, n_i, s_i, mx_i, mn_i)$ represents the number n_i of occurrences of the property l_i in the vertices of G belonging to the cluster c; whereas, s_i, mx_i, mn_i represent, respectively, the sum, the max and the min values that this property assumes. We remark that for an internal cluster c of the hierarchy C, n_i and s_i are computed by means of the corresponding fields in the children of c. Depending on the context, the tuple can be extended with the result of the application of any aggregate function for the maintenance of aggregated data at different granularities.

In this section, we refer to the hierarchy cluster $C = (c_r, V_C, E_C)$ defined on a graph G that presents $L = \{l_1, \ldots, l_k\}$ addictive properties. The `LeafIndex` structure is formally defined as follows.

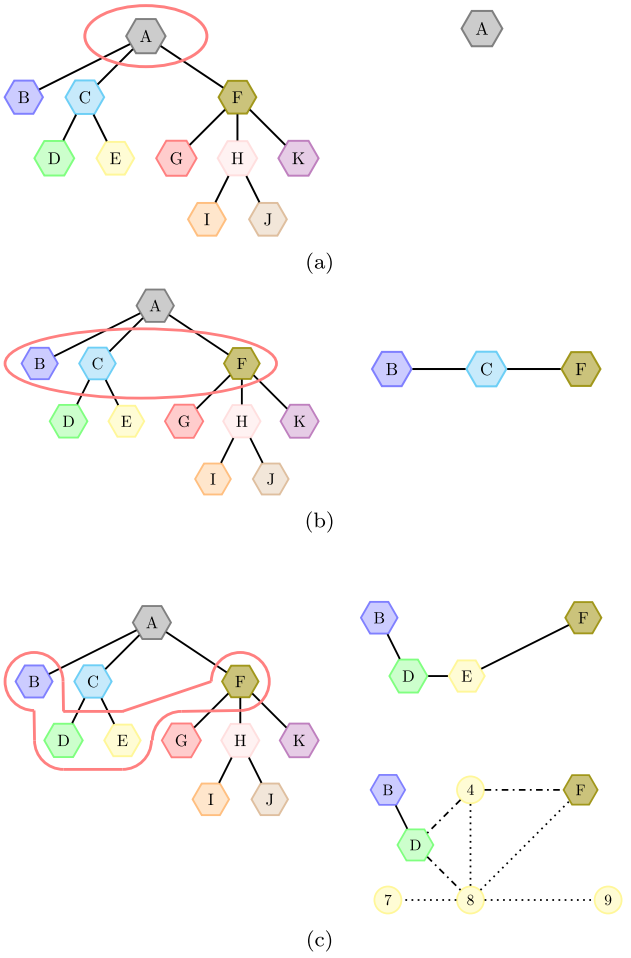


FIGURE 2. Multi-resolution visualizations are obtained from the graph *G* in Figure 1 by considering the cluster hierarchy and the current slice (delimited by a red line). (*a*) the slice consists of the hierarchy root and *G* is represented as a single meta-node. (*b*) the slide consists of the children of the hierarchy root. In this case three meta-nodes are shown along with their relationships. (*c*) the slice involves leaf nodes of the hierarchy. In this case, either meta-nodes (top part) or meta-nodes with atomic-nodes (bottom part) can be shown.

Definition 1 (`LeafIndex`) : The `LeafIndex` *LI* associates a tuple of values to each cluster of C as follows:

- $\forall c_i \in L(V_C)$, $0 \leq i \leq |L(V_C)|-1$, $LI(c_i) = (i, i, lev, n_V, n_E, agg)$, where i is the rank in the list $L(V_C)$, $lev = level(c_i)$, n_V and n_E are the number of vertices/edges of G belonging to c_i, and agg is the set of tuples associated with addictive properties.
- $\forall c \in V_C \setminus L(V_C)$, $LI(c) = (i_{min}, i_{max}, lev, n_V, n_E, agg)$ where $lev = level(c_i)$, $i_{min} = min_{c' \in L_c(V_C)}(LI_1(c'))$, $i_{max} = max_{c' \in L_c(V_C)}(LI_2(c'))$, n_V and n_E are the number of vertices/edges of G belonging to c, and agg is the set of tuples associated with addictive properties computed by the children of c. LI_j denotes the j-component of the 6-tuple. □

Accordingly, each vertex $v \in G$ is associated with a single index i (denoted as $LI_G(v)$), corresponding to the specific leaf cluster in which it has been included (i.e. it implements $\mathcal{F}$), because, for each leaf cluster c_l, $LI_1(c_l) = LI_2(c_l)$. Figure 3 reports the tuples of the *LI* index associated with the clusters of our hierarchy.

TABLE 1. Basic operations on a cluster hierarchy *C*.

operation	computation with the index
$c \in desc(c')$	$LI_1(c) \leq LI_1(c') \leq LI_2(c') \leq LI_2(c)$
$c \in child(c')$	$c \in desc(c') \wedge LI_3(c) = LI_3(c') + 1$
$c \sim c'$	$c \in desc(c') \vee c' \in desc(c)$
$c \not\sim c'$	$LI_2(c) < LI_1(c') \vee LI_2(c') < LI_1(c)$
$v \in c$	$LI_1(c) \leq LI_1(\mathcal{F}(v)) \wedge LI_2(c) \geq LI_2(\mathcal{F}(v))$

Lemma 1: The following properties hold on the LI_1 and LI_2 fields of `LeafIndex` for a cluster hierarchy C.

1) $\forall c \in V_C, LI_1(c) \leq LI_2(c)$.
2) $\forall c \in V_C \setminus L(V_C)$ (internal nodes of C)
   a) $LI_1(c) = min_{c' \in child(c)}(LI_1(c'))$,
   b) $LI_2(c) = max_{c' \in child(c)}(LI_2(c'))$.
3) $\forall c_1, c_2 \in V_C, c_2 \in desc(c_1) \iff LI_1(c_1) \leq LI_1(c_2) \wedge LI_2(c_1) \geq LI_2(c_2)$
4) $\forall c_1, c_2 \in V_C, c_1 \not\sim c_2 \iff LI_2(c_1) < LI_1(c_2) \vee LI_2(c_2) < LI_1(c_1)$
5) $\forall c, c_1, c_2 \in V_C, LI_1(c_1) \neq LI_1(c_2) \vee LI_2(c_1) \neq LI_2(c_2)$, i.e. $(LI_1(c), LI_2(c))$ is unique in V_C.
6) $\forall c_1, c_2 \in V_C$, only one of these alternatives occurs:
   a) $LI_1(c_1) \leq LI_2(c_1) < LI_1(c_2) \leq LI_2(c_2)$ this happens when $c_1 \not\sim c_2$.
   b) $LI_1(c_2) \leq LI_2(c_2) < LI_1(c_1) \leq LI_2(c_1)$ this happens when $c_1 \not\sim c_2$.
   c) $LI_1(c_1) \leq LI_1(c_2) \leq LI_2(c_2) \leq LI_1(c_1)$ this happens when $c_2 \in desc(c_1)$.
   d) $LI_1(c_2) \leq LI_1(c_1) \leq LI_2(c_1) \leq LI_2(c_2)$ this happens when $c_1 \in desc(c_2)$. □

Next Lemma is a nice index property for determining the number of leaf nodes on a subtree rooted in c.

Lemma 2: $\forall c \in V_C$, let $C' = (V', E', c)$ the subtree rooted in c. Then, $|L(V')| = LI_2(c) - LI_1(c) + 1$. □

Corollary 3: $|L(V_C)| = LI_2(c_r) - LI_1(c_r) + 1$. □

Table 1 introduces some basic operations on the cluster tree can be computed by exploiting the `LeafIndex` structure in constant time according to the following theorem. These operations will be exploited in our algorithms. Note that, $v \in c$ in Table 1 is a short notation for $v \in cluster(c)$.

Theorem 4: Let $\mathcal{F} : V \rightarrow L(V_C)$ be the clustering function between a graph G and a cluster hierarchy C, with $c, c' \in V_C$ and $v \in V$. The operations in Table 1 can be computed in constant time by means of `LI`. □

Algorithm 1 reports the code for the generation of the first three fields of the `LeafIndex`. The last fields can be easily computed by aggregating the values from the graph G for leave clusters of C, and for an internal cluster c by aggregating the values occurring in the children of c (this process is not described any longer because it is straightforward). The parameters of the function create_LI

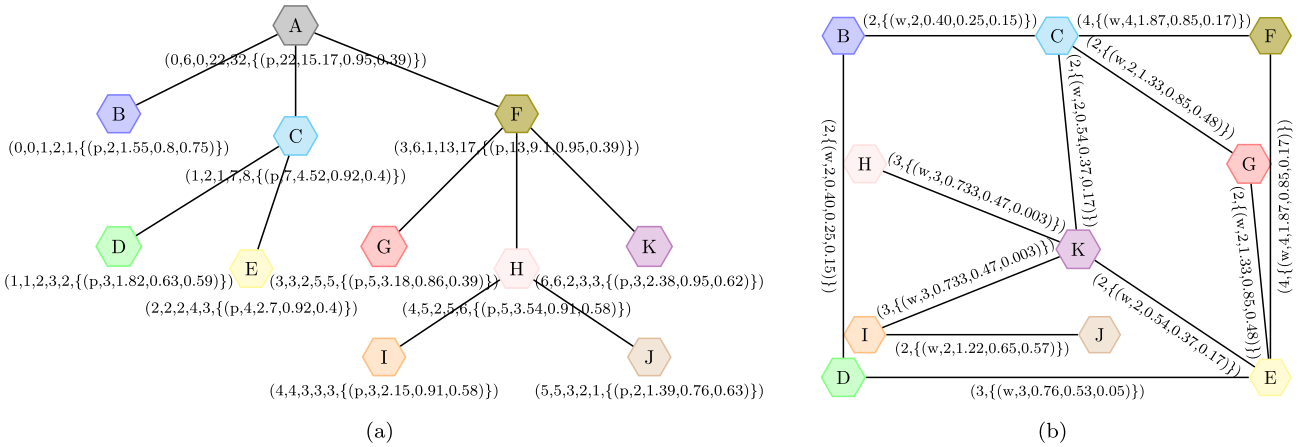


FIGURE 3. LeafIndex and edgeIndex.

are the vertex c, the next LI_1 to be assigned, the current level *lev*, and the `LeafIndex` structure LI. Initially, the function is called on the root vertex c_r, with 0 as first LI_1 and also as level. The function has two different behaviors in the case of a leaf vertex (lines 10 and 11) or of a non-leaf vertex (lines 3 to 8). For the leaf vertex, the function simply assigns the input value (the next LI_1) i to both LI_1 and LI_2 and the level *lev* to the current vertex c and the value i is returned back. In the other case, function create_LI is recursively invoked on each of its children. Since for the first child, the value j to be passed to the recursive call is the same value of the parent node, j is initialized to $i-1$. The other parameters are: the child of position k (c_k), the current value of j incremented by one (the next LI_1, Lemma 1 - Point 2), the next level (lev+1) and the LI index. Note that at each interaction, j is the last LI_2 received in the previous iteration incremented by one (Lemma 8 - Point 1). In lines 7 and 8, the function assigns the LI index to the current vertex c and returns the same value. The level of all children of c is $lev+1$.

Theorem 5: Algorithm 1 computes LI in $\mathcal{O}(|V_C|)$. □

Algorithm 1 `CreateLeafIndex`

Input: The cluster tree $C = (c_r, V_C, E_C)$
Output: the index `LI`
$LI \leftarrow \{\}$
create_LI $(c_r, 0, 0, LI)$
return LI

```
1:  function create_LI(c, i, lev, LI)
2:      if |child(c)| > 0 then          ▷ child(c) = c_1, ..., c_n
3:          j = i − 1;
4:          for k = 1 to n do
5:              j = create_LI(c_k, j + 1, lev + 1, LI)
6:          end for
7:          LI(c) = (i, j, lev)
8:          return j
9:      else                            ▷ c is a leaf node
10:         LI(c) = (i, i, lev)
11:         return i
12:     end if
13: end function
```

D. EDGEINDEX STRUCTURE

An index can be also created on the edges among non-overlapping clusters in C by means of the edges E of G. This index (named EI) is a graph (V_I, E_I, ϕ_I) whose nodes are the clusters in V_C and E_I contains the edges between two non-overlapping clusters (c_1, c_2) for which at least an edge exists in G among the nodes belonging to the clusters c_1 and c_2. Since addictive properties can be also specified on the edges of the graph G, the pair $(n_E, \{(l_1, n_1, s_1, mx_1, mn_1), \ldots, (l_k, n_k, s_k, mx_k, mn_k)\})$ can be associated with each edge of E_I through function ϕ_I, where n_E is the number of edges in G across c_1 and c_2 and the tuples correspond to the addictive properties $l_1, \ldots, l_k$ on edges across the two clusters. When (c_1, c_2) is an edge between leaves of C, in the tuple $(l_i, n_i, s_i, mx_i, mn_i)$ of an addictive property l_i,

- n_i is the number of edges in G among the clusters c_1 and c_2 presenting the property l_i;
- s_i is the sum of the values associated with l_i;
- mx_i, mn_i are the max and min value associated to l_i in the edges among the clusters c_1 and c_2.

When c_1 or c_2 is not a leaf cluster, n_i and s_i are computed as the sum of the corresponding fields in the edges of the set $\{(c, c') | c \in child(c_1), c' \in child(c_2) \wedge (c, c') \in E_I\}$, and mx_i and mn_i are the max and the min of the corresponding fields. This is an important observation for reducing the cost of computation of these tuples. Moreover, during the exploration of the graph, aggregate properties that hold among clusters and their aggregated strength do not need to be computed on the fly.

Definition 2 (`EdgeIndex`*):* The `EdgeIndex` EI is a graph (V_I, E_I, ϕ_I), where $V_I = V_C \backslash \{c_r\}, E_I = \{(c_1, c_2) \in E_C$

Algorithm 2 `CreateEdgeIndex`

Input: The cluster tree $C = (c_r, V_C, E_C)$,
The graph $G = (V, E, \eta, \xi)$
the set L of addictive properties
Output: the index $EI = (V_I, E_I, \phi_I)$
$V_I \leftarrow V_C \setminus \{c_r\}$
$E_I \leftarrow \emptyset, \bar{E}_I \leftarrow \emptyset$
ϕ_I empty function
1: **for each** $(v_1, v_2) \in E$ s.t. $\mathcal{F}(v_1) \neq \mathcal{F}(v_2)$ **do**
2: includeEdge$(L, (v_1, v_2), \bar{E}_I, (\mathcal{F}(v_1), \mathcal{F}(v_2)), \phi_I)$
3: **end for**
4: **for each** $(\bar{c_1}, \bar{c_2}) \in \bar{E}_I$ **do**
5: **for each** $(c_1, c_2) \in path(\bar{c_1}) \times path(\bar{c_2})$
6: s.t. $c_1 \not\sim c_2 \wedge (c_1 \neq \bar{c_1} \vee c_2 \neq \bar{c_2})$ **do**
7: includeEdge$(L, (c_1, c_2), E_I, (\bar{c_1}, \bar{c_2}), \phi_I)$
8: **end for**
9: **end for**
10: **return** $(V_I, E_I \cup \bar{E}_I, \phi_I)$

11: **procedure** includeEdge$(L, e_1, E_{var}, e_2, \phi)$
12: **if** $e_2 \in E_{var}$ **then**
13: **for each** $l \in L$ addictive property on e_2 **do**
14: $\phi_I(e_2).n = \phi_I(e_2).n + 1$
15: $\phi_I(e_2).s = \phi_I(e_2).s + e_1.s$
16: $\phi_I(e_2).mx = max(\phi_I(e_2).mx, e_1.mx)$
17: $\phi_I(e_2).mn = min(\phi_I(e_2).mn, e_1.mn)$
18: **end for**
19: **else**
20: $E_{var} = E_{var} \cup \{e_2\}$
21: **for each** $l \in L$ addictive property on e_1 **do**
22: let $(l, 1, val, val, val)$ a tuple for l ▷ val, value of l
23: $\phi(e_2) = \phi(e_2) \cup \{(l, 1, val, val, val)\}$
24: **end for**
25: **end if**
26: **end procedure**

s.t. $c_1 \not\sim c_2$, and $\exists (v_1, v_2) \in E$ s.t. $cluster(v_1) \in desc(c_1)$ and $cluster(v_2) \in desc(c_2)\}$, and $\forall (c_1, c_2) \in E_I$, $\phi_I((c_1, c_2)) = (n_E, agg)$, where n_E is the number of edges in G between the clusters c_1 and c_2, and agg is the set of addictive properties as discussed above. □

The construction of this index is realized in two steps by means of Algorithm 2 (in this case we report the construction of the tuples for addictive properties that are generated according to the previous lemma). In the first step (lines between 1 and 3), the edges that occur among the clusters in V_C that are leaves of the hierarchy are determined and included in $\bar{E}_I$, by calling the IncludeEdge procedure. By means of this procedure, we check if the meta-edge e_2 is already included in the set $\bar{E}_I$. In the positive case, the addictive properties L associated with e_1 are included in e_2. In the negative case, the edge e_2 is included in $\bar{E}_I$ and the addictive properties of e_1 used for initializing the aggregated values associated with e_2 by means of the ϕ function. The inclusion of edges in $\bar{E}_I$ is accomplished by selecting the edges whose source and target vertices fall in different leaf clusters. All the leaf nodes of the hierarchy C should be considered because they are all disjoint. Note that this step requires to access the graph G because we need to determine the edges connecting nodes among clusters. Starting from these edges, other edges are included in the index by considering their inclusion relationship with previously calculated edges (lines between 4 and 9). This computation does not require to access the graph G and can be calculated by considering the edges in $\bar{E}_I$ and aggregating them. Specifically, for each pair $(\bar{c_1}, \bar{c_2}) \in \bar{E}_I$ (i.e. leaf vertexes in V_C already determined), we determine pairs (c_1, c_2) of their ancestors for which the following conditions hold:

- c_1, c_2 are respectively ancestor or equal to $\bar{c_1}, \bar{c_2}$
- $c_1 \not\sim c_2$, i.e. not common ancestor of $\bar{c_1}$,$\bar{c_2}$
- $\bar{c_1}, \bar{c_2}$ not both equal respectively to c_1 and c_2

In this case, (c_1, c_2) is included in E_I through the IncludeEdge procedure and the associated addictive properties are kept updated. In IncludeEdge, the evaluation of the previous condition is needed because a pair c_1, c_2 can be the ancestor of other pairs of leaf vertexes $\bar{c_1}, \bar{c_2}$. At line 10, the graph $(V_I, E_I \cup \bar{E}_I)$ (obtained through the union of the edges computed in the two steps) is returned with the calculated properties.

Theorem 6: Algorithm 2 correctly computes the `EdgeIndex` EI in time $\mathcal{O}(|E| + |V_C|^2)$. □

The cardinality of the edges in EI strictly depends on the inter-relationships existing among the cluster of nodes identified in the graph G. So, it is relevant to evaluate its size. The following theorem introduces a finer upper ground limit on this number.

Theorem 7: Let (V_I, E_I, ϕ_I) be the `EdgeIndex`.

$$|E_I| \leq \frac{1}{2}(|V_I| + 1)|V_I| - \sum_{e \in V_I} level(v)$$ □

III. MULTI-RESOLUTION VISUALIZATION AND SLICES

To identify the correct multi-resolution visualization of the graph we introduce the concept of *slice* as a subset of vertices of V_C that are reported in the current view.

A. NODE CHAINS AND SLICES

Starting from the cluster hierarchy structure it is possible to identify sequences of vertices, named *node chain*, that are contiguous according to the `leafindex` structure that we have introduced in the previous section. These vertices are those that can appear in a multi-resolution visualization of G.

Definition 3 (Node Chain): A node chain in a cluster tree C is a non-empty list of vertices $(c_1, \ldots, c_k)$ belonging to V_C s.t. $\forall i (2 < i \leq k), LI_2(c_{i-1}) + 1 = LI_1(c_i)$. □

Example 2: Consider the `LeafIndex` in Figure 3. The following sequences, with their corresponding LI_1 and LI_2 values, are node chains: $(G_{(3,3)}, H_{(4,5)}, K_{(6,6)})$, $(B_{(0,0)}, C_{(1,2)}, G_{(3,3)}, I_{(4,4)})$. □

Lemma 8: The following properties hold on node chains.

1) $\forall c \in V_C \setminus L(V_C)$ (internal node), $child(c)$ is a chain.
2) if $(c_1, .., c_j, .., c_k)$ is a node chain, then $(c_1, .., c_j^1, .., c_j^h .., c_k)$ is a node chain, where $(c_j^1 \ldots c_j^h) = child(c_j)$.
3) if $(c_1, .., c_j^1, .., c_j^h, .., c_k)$ is a node chain, and $child(c_j) = (c_j^1 \ldots c_j^h)$, then $(c_1, \ldots, c_j, \ldots, c_k)$ is a node chain.
4) if $(c_1, .., c_j)$, $(c_h, .., c_k)$ are node chains and $LI_2(c_j) = LI_1(c_h) - 1$, then $(c_1, .., c_k)$ is a node chain.
5) if c_1, c_2 are distinct in a chain, then $c_1 \not\sim c_2$.
6) $\forall c \in V_C \setminus L(V_C)$, $desc(c) \cap L(V_C)$ is a node chain $(c1, .., c_k)$ s.t. $LI_1(c_1) = LI_1(c) \wedge LI_2(c_k) = LI_1(c)$.
7) if $(c_1, \ldots, c_j, \ldots, c_k)$ and $(c_j^1, \ldots, c_j^h)$ are node chains s.t. $LI_1(c_j) = LI_1(c_j^1) \wedge LI_2(c_j) = LI_2(c_j^h)$, then $(c_1, \ldots, c_j^1, \ldots, c_j^h, \ldots, c_k)$ is a node chain.
8) if $cc = (c_1, .., c_h, .., c_k, .., c_j)$ is a node chain, $c \notin cc$, and $cc \cap desc(c) = (c_h, .., c_k)$ is not empty, then $(c_1, .., c, .., c_j)$ is a node chain. □

Node chains that appear in a multi-resolution visualization of the graph G are named *slices*. A slice is a complete chain, that is a chain that cut horizontally the cluster hierarchy in two parts.

Definition 4 (Slice): A *slice* is a *node chain* $(c_1, \ldots, c_k)$ s.t. $\texttt{LI}_1(c_1) = 0$ and $\texttt{LI}_2(c_k) = LI_2(c_r)$, □

Example 3: Figure 2 reports some examples of slices. Figure 2(a) shows a slice in which only the root vertex forms the slice. Figure 2(b) reports a slice corresponding to the explosion of the meta-node A. Figure 2(c) contains a slice obtained by the previous one by exploding the vertex C. The chain formed by $L(V_C)$ is also a slice. □

In the following, $\mathcal{S}(C)$ denotes the set of all *slices* that can be generated on a cluster C, and $S(c)$ the slices of the subtree rooted at $c \in V_C$.

Starting from Lemma 8 (Item 5) that establishes a relationship between slice and disjoint nodes in the cluster hierarchy, the next theorem makes more evident this relationship by claiming that each subset of disjoint vertices can be used in a slice. For instance, in the hierarchy in Figure 3, if we consider the disjoint nodes B, E, H, a slice exists containing them (e.g. the slice (B, D, E, G, H, K)). Since EI represents all possible relationships among disjoint vertices, it can be exploited for the verification of the condition of this theorem.

Theorem 9: $\forall U \subseteq V_C$ s.t. $\forall u_1, u_2 \in U, u_1 \not\sim u_2$, $\exists$ a slice $s \in S(C)$ s.t. $U \subseteq s$. □

The following theorem establishes the cardinality of $S(C)$ by considering the cluster hierarchy structure.

Theorem 10: $|S(C)| = 1 + \prod_{c \in child(root(C))} |S(c)|$ □

Example 4: Consider the hierarchy C in Figure 1. $S(C) = \{(A), (B, C, D), (B, D, E, F), (B, C, G, H, K), (B, D, E, G, H, K), (B, C, G, I, J, K), (B, D, E, G, I, J, K)\}$. Therefore $|S(C)| = 7$. The same result can be obtained by considering the subtrees rooted at the children of the root of C. Indeed, $S(subtree(B)) = \{(B)\}$, $S(subtree(C)) = \{(C), (D, E)\}$, $S(subtree(F)) = \{(F), (G, H, K), (G, I, J, K)\}$. Thus, $|S(C)| = 1 + |S(subtree(B))| * |S(subtree(C))| * |S(subtree(F))| = 1 + 1 * 2 * 3 = 7$. □

B. MULTI-RESOLUTION VISUALIZATION

A slice represents a resolution level according to which the graph G should be visualized and the current visualization can contain atomic-nodes of G and meta-nodes. For this reason, we introduce the concept of *expanded slice* for representing the expanded meta-nodes in the current visualization.

Definition 5 (Expanded Slice): An expanded slice is a pair (cc, σ) where cc is a slice according to Definition 4 and $\sigma : L(V_C) \cap cc \rightarrow Boolean$ is a partial predicate indicating where the leaf cluster is expanded. □

For the sake of simplicity, in the remainder, we consider $\sigma(c)$ undefined equivalent to $\sigma(c) = \texttt{false}$. This assumption simplifies the presentation of our results.

Example 5: The two multi-resolution visualizations on the right part of Figure 2(c) are obtained by the slice reported in the left part of the figure. In the first case, the leaf nodes are not expanded (i.e. σ is undefined or false for B, D, E, F). By contrast, in the other the leaf node E is expanded (i.e. $\sigma(E) = \texttt{true}$). □

Having introduced the expanded slice, we are now ready to introduce the multi-resolution visualization of a graph G that can be obtained by an extended slice. Intuitively, a multi-resolution visualization of a graph G is a graph G_L with two kinds of vertices: *meta-nodes*, corresponding to the clusters present in the slice that have not been expanded, and *atomic-nodes* representing the vertices occurring in the graph G that are instances of the clusters belonging to the slice that have been expanded. Due to the different types of nodes that are present in G_L, three kinds of edges can be identified in E_L: edges of the original graph G (atomic-edges); those that represent relationships among (non-expanded) clusters (meta-edges); and finally, those that represent relationships among atomic-nodes and (non-expanded) clusters (mix-edges).

Definition 6 (Multi-Resolution Visualization): Consider a cluster tree $C = (c_r, V_C, E_C)$ built on a graph $G = (V, E, \eta, \xi)$, LI and $\texttt{EI}=(V_I, E_I, \phi_I)$ the indexing structures introduced in the previous section, and (cc, σ) an expanded slice indicating the leaf clusters that are marked as expanded (where $cc = (c_1, \ldots, c_k)$). A multi-resolution visualization for G according to (cc, σ), is a graph $G_L = (V_L, E_L, \eta_L, \xi_L)$, in which two kinds of vertices ($V_L = V_L^{\texttt{meta}} \cup V_L^{\texttt{node}}$) can be identified:

- $V_L^{\texttt{meta}} = \{c | c \in cc, \sigma(c) = \perp\}$
- $V_L^{\texttt{node}} = \{v \in V | \mathcal{F}(v) \in \{c \in cc | \sigma(c) = \texttt{true}\}\}$

and three kinds of edges can be highlighted:

- $E_L^{\texttt{meta}} = \{(c, \bar{c}) | c, \bar{c} \in V_L^{\texttt{meta}}, (c, \bar{c}) \in E_I\}$
- $E_L^{\texttt{node}} = \{(v, \bar{v}) \in E | v, \bar{v} \in V_L^{\texttt{node}}\}$
- $E_L^{\texttt{mix}} = \{(v, c) | v \in V_L^{\texttt{node}}, c \in V_L^{\texttt{meta}} \wedge (\mathcal{F}(v), c) \in E_I\}$

η_L and ξ_L are computed relying on the corresponding ones defined on G and on LI and EI. □

Example 6: Consider the expanded slice $((B, D, E, F), \phi)$, where $\phi(E) = \texttt{true}$. In this case, the visualization contains $V_L^{\texttt{meta}} = \{B, D, F\}$, $V_L^{\texttt{node}} = \{4, 7, 8, 9\}$, $E_L^{\texttt{meta}} = \{(B, D)\}$, $E_L^{\texttt{mix}} = \{(D, 4), (D, 8), (f, 4), (F, 8)\}$, and $E_L^{\texttt{node}} = \{(4, 8), (7, 8), (8, 9)\}$. The graphical representation is in the right bottom part of Figure 2(c). □

The next lemma represents a link between the EdgeIndex and the multi-resolution visualization that can be used for improving the algorithms' performances.

Lemma 11: The following properties hold on $G_L = (V_L, E_L, \eta_L, \xi_L)$ and its $\texttt{EI} = (V_I, E_I, \phi_I)$:

1) $V_L^{\texttt{meta}} \subseteq V_I$,
2) $E_L^{\texttt{meta}} \subseteq E_I$,
3) $\forall (c_1, c_2) \in E_L^{\texttt{meta}}, \xi_L((c_1, c_2)) = \phi_I((c_1, c_2))$ □

IV. OPERATIONS ON THE VISUALIZATION

The initial multi-resolution visualization of a graph G according to a cluster hierarchy C consists of a single meta-node (the root c_r of C) and no edges are present (i.e. $G_L = (\{c_r\}, \emptyset, \eta_L, \emptyset)$, where $\eta_L(c_r) = \texttt{LI}(c_r)$).

Starting from that, the operations introduced in this section can be applied for changing the visualization. All the operations are specified starting from an expanded slide and lead to a newly expanded slice. Even if the visualization can always start from the initial graph G, in the section we provide incremental algorithms that allow us to obtain the visualization that results from the application of the operation on the current visualization. Note that this incremental approach is fundamental for a local application of the modifications to the part of the graph visualization that needs to be changed.

A. THE $\texttt{zoom}_{\texttt{IN}}$ OPERATION

The $\texttt{zoom}_{\texttt{IN}}$ operation can be invoked on the meta-node c for expanding the visualization at a deeper resolution level. When c is a leaf of the cluster hierarchy, the expansion introduces the vertices in G belonging to the cluster c. This action is simply marked in the slice.

Definition 7 ($\texttt{zoom}_{\texttt{IN}}$ Operation): Let G_L be a multi-resolution visualization of a graph G according to an expanded slice $ecc = ((c_1, .., c_j, .., c_k), \sigma)$ and a cluster hierarchy C. The $\texttt{zoom}_{\texttt{IN}}$ operation on the meta-node c_j ($\sigma(c_j) \neq \texttt{true}$) leads to a new expanded slice as follows

$$ecc' = \begin{cases} ((c_1, .., c_j^1, ..c_j^n, .., c_k), \sigma) & \text{if } c_j \notin L(V_C) \\ & \{c_j^1, .., c_j^n\} = child(c_j) \\ ((c_1, .., c_j, .., c_k), \sigma[\texttt{true}/c_j]) & \text{if } c_j \in L(V_C) \end{cases}$$

where: $\sigma[\texttt{true}/c_j]$ indicates function σ in which only the value for c_j is changed in true. Then, a new visualization $G_L^{\texttt{new}}$ is obtained. □

Example 7: The right part of Figure 2(a) shows the initial graph visualization according to the slice highlighted in the cluster hierarchy (left part of the figure). Then, the right part of Figure 2(b) shows the result of the application of the $\texttt{zoom}_{\texttt{IN}}$ operation on the meta-node A. Finally, the top right part of Figure 2(c) shows the result of the application of $\texttt{zoom}_{\texttt{IN}}$ on the meta-node C, which also allows the visualization of atomic-nodes (as shown in the bottom right part of Figure 2(c)). □

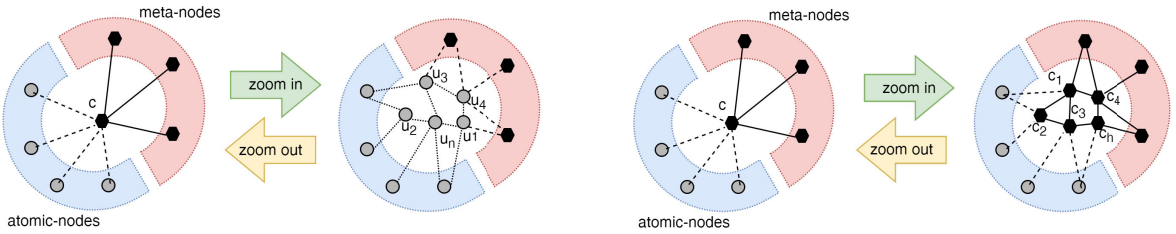
Algorithm 3 $\texttt{zoom}_{\texttt{IN}}$

Input: The cluster tree $C = (c_r, V_C, E_C)$,
The graph $G = (V, E, \eta, \xi)$
the set L of addictive properties
the indexing structures LI and $EI = (V_I, E_I, \phi_I)$
the expanded slice (cc, σ)
a node $c \in cc : \sigma(c) \neq \texttt{true}$
the current visualization $G_L = (V_L, E_L, \eta_L, \xi_L)$
Output: a new visualization $G_L^{\texttt{new}} = (V_L^{\texttt{new}}, E_L^{\texttt{new}}, \eta_L^{\texttt{new}}, \xi_L^{\texttt{new}})$

$V_L^{\texttt{node}}(c) \leftarrow \{v | v \in V_L^{\texttt{node}} \wedge (v, c) \in E_L\}$
$V_L^{\texttt{meta}}(c) \leftarrow \{v | v \in V_L^{\texttt{meta}} \wedge (v, c) \in E_L\}$
$E_L^{\texttt{new}} \leftarrow \emptyset$
if c is a leaf cluster in C **then**
  $V(c) \leftarrow cluster(c)$
  $E(c) \leftarrow \{(v_1, v_2) | (v_1, v_2) \in E \wedge v_1, v_2 \in V(c)\}$
  $\xi_L^{\texttt{new}} = \{\phi(e) | e \in E(c)\}$
  for each $(v_1, v_2) \in E : \mathcal{F}(v_1) = c \wedge \mathcal{F}(v_2) \neq c$ **do**
    if $v_2 \in V_L^{\texttt{node}}(c)$ **then**
      includeEdge$(L, (v_1, v_2), E_L^{\texttt{new}}, (v_1, v_2), \xi_L^{\texttt{new}})$
    else
      Let $c_t \in cc$ s.t. $v_2 \in c_t$
      includeEdge$(L, (v_1, v_2), E_L^{\texttt{new}}, (v_1, c_t), \xi_L^{\texttt{new}})$
    end if
  end for
else
  $V(c) \leftarrow child(c)$
  $E(c) \leftarrow \{(c_1, c_2) \in E_I | c_1, c_2 \in V(c)\}$
  $\xi_L^{\texttt{new}} = \{\phi_I(e) | e \in E(c)\}$
  for each $(v_1, v_2) \in E$ s.t. $\exists c_t \in V(c)$ s.t. $v_1 \in c_t$
    and $v_2 \in V_L^{\texttt{node}}(c)$ **do**
    includeEdge$(L, (v_1, v_2), E_L^{\texttt{new}}, (c_t, v_2), \xi_L^{\texttt{new}})$
  end for
  $E_L^{\texttt{new}} = E_L^{\texttt{new}} \cup \{(c_1, c_2) \in E_I | c_1 \in V(c), c_2 \in V_L^{\texttt{meta}}(c)\}$
  $\xi_L^{\texttt{new}} = \xi_L^{\texttt{new}} \cup \{\phi_I(c_1, c_2) | c_1 \in V(c), c_2 \in V_L^{\texttt{meta}}(c)\}$
end if
$V_L^{\texttt{new}} = (V_L \setminus \{c\}) \cup V(c)$
$E_L^{\texttt{new}} = (E_L \setminus \{(v, c) | (v, c) \in E_L\}) \cup E_L^{\texttt{new}} \cup E(c)$
$\xi_L^{\texttt{new}} = (\xi_L \setminus \{\xi_L((v, c)) | (v, c) \in E_L\}) \cup \xi_L^{\texttt{new}}$
$\eta_L^{\texttt{new}} = (\eta_L \setminus \{\eta_L(c)\}) \cup \{\eta(v) | v \in V(c)\}$
return $(V_L^{\texttt{new}}, E_L^{\texttt{new}}, \eta_L^{\texttt{new}}, \xi_L^{\texttt{new}})$

Algorithm 3 presents the steps executed when the $\texttt{zoom}_{\texttt{IN}}$ operation is invoked on a vertex c of the current visualization G_L. The current expanded slice (cc, σ) should contain c and $\sigma(c) \neq \texttt{true}$ (indeed, only meta-nodes can be expanded). As depicted in the left part of Figure 4(a), the meta-node c in G_L can be connected with: other meta-nodes (named $V_L^{\texttt{meta}}(c)$ and depicted in the red area) by exploiting meta-edges; or, with atomic-nodes (named $V_L^{\texttt{node}}(c)$ and depicted in the blu area) by exploiting mix-edges. The effect of the $\texttt{zoom}_{\texttt{IN}}$ operation is the removal of c (and its incident edges) and the introduction of new nodes and edges that should be connected with the nodes in the colored areas.

When c is a leaf node of the cluster hierarchy C (left part of Figure 4(a)), c needs to be substituted with its atomic-nodes (as in the right part of the figure). Therefore, the atomic-nodes of cluster c (i.e. $V(c) = \{u_1, u_2, \ldots, u_n\}$), all the related



(a) leaf cluster c with atomic-nodes $u_1, ..., u_n$

(b) internal cluster c with $c_1, \ldots, c_h$ child clusters

FIGURE 4. Main cases in the execution of $\texttt{zoom}_{\texttt{IN}}$ and $\texttt{zoom}_{\texttt{OUT}}$ operations.

internal edges (i.e. $E(c)$), and properties are determined (lines 5 to 7) that correspond to the graph delimited by the two colored areas on the right-hand side of Figure 4(a). At this point, the nodes in $V(c)$ should be connected with nodes in the colored areas. For the atomic-nodes in the blu area, we need to identify edges $(v_1, v_2) \in E$ such that $v_1 \in V(c)$ and $v_2 \in V_L^{\text{node}}(c)$. When this condition is verified the atomic edge (v_1, v_2) can be included in the new visualization. For the meta-nodes in the red area, we need to identify edges $(v_1, v_2) \in G$ such that $v_1 \in V(c)$ and v_2 belongs to a cluster c_t of the current extended slice. When this condition holds, the mix-edge (v_1, c_t) can be included in G_L^{new}. In both cases addictive properties of the included edges are properly updated.

When c is an internal node of C (see the left-hand side of Figure 4(b)), c needs to be substituted with its child meta-nodes. Therefore, similarly to what has been done in the previous case, the children $c_1, \ldots, c_h$ of c are determined (line 17) along with the edges $E(c)$ between nodes in $V(c)$ (by exploiting the E_I index, line 18), and the related node and edge properties taken from EI and LI (line 19). The obtained sub-graph corresponds to the nodes and edges delimited by the two colored areas on the right-hand side of Figure 4(b). We have now to determine the edges that connect nodes in $V(c)$ with those contained in the two colored areas. For the nodes in the blu area, we need to identify the edges $(v_1, v_2) \in E$ s.t. v_1 belongs to a cluster c_t among those contained in $V(c)$ and v_2 belongs to the nodes in the blue area (i.e. $v_2 \in V_L^{\text{node}}(c)$). When this condition holds, the edge (c_t, v_2) can be included in the new visualization. For the nodes in the red area, by exploiting the index $\texttt{EI}$, it is easy to identify the edges existing between $V(c)$ and those contained in the red area (line 24). Line 27 creates the final set V_L^{new} from the previous V_L removing the node c and adding the ones in $V(c)$; similarly, at line 28 the new E_L^{new} is generated by removing from E_L edges involving c and including the edges previously computed. Finally, node/edge labeling functions are updated accordingly (lines 29-30).

B. THE $\texttt{zoom}_{\texttt{OUT}}$ OPERATION

The $\texttt{zoom}_{\texttt{OUT}}$ operation is the inverse operation of $\texttt{zoom}_{\texttt{IN}}$ and allows the visualization of the graph at a lower resolution level. Differently from $\texttt{zoom}_{\texttt{IN}}$, this operation can be invoked both on a meta-node or on an atomic-node. In the last case, the slice does not change (only the associated σ function is updated).

Definition 8 ($\texttt{zoom}_{\texttt{OUT}}$ Operation): Let G_L be a multi-visualization of a graph G according to an expanded slice $ecc = ((c_1, .., c_j^1, .., c_j^n, .., c_k), \sigma)$ and a cluster hierarchy C. The application of the $\texttt{zoom}_{\texttt{OUT}}$ operation on:

1) a meta-node c_j^p $(1 \leq p \leq n, c_j^p \neq root(C), \sigma(c_j^p) = \texttt{false})$ leads to the expanded slice $ecc' = ((c_1, .., c_j, .., c_k), \sigma)$ with $c_j = parent(c_j^p)$ and $\{c_j^1, .., c_j^n\} = ecc \cap desc(c_j)$.
2) an atomic-node $v \in G$ s.t. $\mathcal{F}(v) = c_j^p$ leads to $ecc' = ((c_1, .., c_j^1, .., c_j^n, .., c_k), \sigma[\texttt{false}/c_j^p])$.

From ecc', a new visualization G_L^{new} is obtained. □

Example 8: The invocation of $\texttt{zoom}_{\texttt{OUT}}$ on any one of the atomic-nodes in the right bottom part of Figure 2(c) leads to the graph visualization reported in the top part of Figure 2(c). In this case, the slice does not change. The invocation of $\texttt{zoom}_{\texttt{OUT}}$ on $\texttt{D}$ leads to the graph visualization on the right part of Figure 2(b). □

Algorithm 4 reports the pseudo-code of the $\texttt{zoom}_{\texttt{OUT}}$ operation. This operation is simpler than the $\texttt{zoom}_{\texttt{IN}}$ operation because it works only on the current multi-resolution visualization G_L by taking into account the cluster hierarchy C and the node v that needs to be collapsed. After the initialization of the data structures, we need to identify the nodes to be eliminated from the current visualization (i.e. $V(c)$) where c is the meta-node that is determined according to the node v on which the $\texttt{zoom}_{\texttt{OUT}}$ is invoked. If v is an atomic-node (e.g. one of the $u_1, \ldots, u_n$ nodes in the right-hand side of Figure 4(a)), c is the leaf cluster associated with it, and $V(c) = \{u_1, \ldots, u_n\}$ because we need to remove all the atomic-nodes of the cluster c. If v is a meta-node, c is its parent in the hierarchy and $V(c)$ contains the meta-nodes c' in the expanded slice ecc such that $c' \in desc(c)$ and $\sigma(c') = false$ (meta-nodes in G_L that are descendant of c) or atomic-nodes v of a cluster $\bar{c}$ in the expanded slice ecc such that $\sigma(\bar{c}) = true$ (the picture in the right-hand side of Figure 4(b) shows the case in which only meta-nodes are children of c). Then (lines 10-12) all new edges in which

Algorithm 4 $\texttt{zoom}_{\texttt{OUT}}$

Input: The cluster tree $C = (c_r, V_C, E_C)$,
the set L of addictive properties
the expanded slice (cc, σ)
the current visualization $G_L = (V_L, E_L, \eta_L, \xi_L)$
a node $v \in V_L$
Output: a new visualization $G_L^{new} = (V_L^{new}, E_L^{new}, \eta_L^{new}, \xi_L^{new})$

1: $(E_L^{new}, \xi_L^{new}) \leftarrow (\emptyset, \emptyset)$
2: **if** $v \in V_L^{node}$ **then**
3: $c \leftarrow \mathcal{F}(v)$
4: $V(c) \leftarrow cluster(c)$
5: **else**
6: $c \leftarrow parent(v)$
7: $V(c) \leftarrow \{c' \in V_L^{meta} | c' \in desc(c)\} \cup$
8: $\{v' \in V_L^{node} | v' \in c\}$
9: **end if**
10: **for each** $(v_1, v') \in (V_L \setminus V(c)) \times V(c)$ s.t. $(v_1, v') \in E_L$ **do**
11: includeEdge$(L, (v_1, v'), E_L^{new}, (c, v_1), \xi_L^{new})$
12: **end for**
13: $V_L^{new} = (V_L \cup \{c\}) \setminus V(c)$
14: $E_L^{new} = E_L^{new} \cup (E_L \setminus \{(v_1, v_2) \in E_L | v_1 \in V(c) \vee v_2 \in V(c)\})$
15: $\xi_L^{new} = \xi_L^{new} \cup (\xi_L \setminus \{\xi_L((v_1, v_2)) | v_1 \in V(c) \vee v_2 \in V(c)\})$
16: $\eta_L^{new} = (\eta_L \setminus \{\eta_L(c_1) | v_1 \in V(c)\}) \cup \{\eta(c)\}$
17: **return** $(V_L^{new}, E_L^{new}, \eta_L^{new}, \xi_L^{new})$

one of its vertex falls in the colored area are substituted with an edge that connects the node in the colored area and c. By exploiting the sets previously described, the algorithm provides the components of G_L^{new} (lines 13-16) by inserting the new elements and removing the old ones, and this leads to the situation depicted in the left-hand side of Figure 4(b).

C. THE $\texttt{zoom}_{\texttt{DEEP}}$ OPERATION

The $\texttt{zoom}_{\texttt{DEEP}}$ operation allows the expansion of a cluster at any level of the hierarchy and reports its atomic-nodes. The same effect can be realized by subsequent $\texttt{zoom}_{\texttt{IN}}$ invocations, but this operation is faster.

Definition 9 ($\texttt{zoom}_{\texttt{DEEP}}$ Operation): Let G_L be a multi-resolution visualization of a graph G according to an expanded slice $ecc = ((c_1, .., c_j, .., c_k), \sigma)$ and a cluster hierarchy C. The $\texttt{zoom}_{\texttt{DEEP}}$ operation on a meta-node c_j ($\sigma(c_j) \neq \texttt{true}$) allows to obtain a new expanded slice:

$$ecc' = \begin{cases} ((c_1, .., c_j^1, ..c_j^n & \text{if } c_j \notin L(C) \\ \quad .., c_k), \sigma') & \{c_j^1, ..c_j^n\} = desc(c_j) \cap L(C) \\ & \sigma' = \sigma[\texttt{true}/c_j^1]..[\texttt{true}/c_j^n] \\ ((c_1, .., c_j, .., c_k), & \text{if } c_j \in L(C) \\ \quad \sigma[\texttt{true}/c_j]) & \end{cases}$$ □

Example 9: Starting from the slice in Figure 2 b) and applying $\texttt{zoom}_{\texttt{DEEP}}$ to C, the slice and the visualization in Figure 5 are obtained. The graph has both meta-nodes (B and F) and atomic-nodes (instances of D and E). □

Algorithm 5 is a slight extension of the treatment of a leaf meta-node of Algorithm 3. The main difference consists in the computation of the $V(c)$ and $E(c)$ sets. In this case, they need to contain the atomic-nodes (and corresponding edges) of the cluster c (independently from the fact that c is a leaf or internal cluster). Having said that, the other

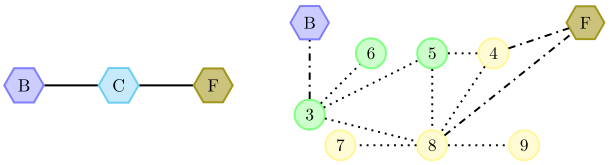


FIGURE 5. Multi-resolution visualization obtained by applying $\texttt{zoom}_{\texttt{DEEP}}$ to the meta-node C of the slice in Figure 2 b).

Algorithm 5 $\texttt{zoom}_{\texttt{DEEP}}$

Input: The cluster tree $C = (c_r, V_C, E_C)$,
The graph $G = (V, E, \eta, \xi)$
the set L of addictive properties
the indexing structures LI and $EI = (V_I, E_I, \phi_I)$
the expanded slice (cc, σ)
a node $c \in cc : \sigma(c) \neq \texttt{true}$
the current visualization $G_L = (V_L, E_L, \eta_L, \xi_L)$
Output: a new visualization $G_L^{new} = (V_L^{new}, E_L^{new}, \eta_L^{new}, \xi_L^{new})$

1: $V_L^{node}(c) \leftarrow \{v | v \in V_L^{node} \wedge (v, c) \in E_L\}$
2: $V_L^{meta}(c) \leftarrow \{v | v \in V_L^{meta} \wedge (v, c) \in E_L\}$
3: $E_L^{new} \leftarrow \emptyset$
4: $V(c) \leftarrow \{v \in V | LI_1(c) \leq LI_G(v) \leq LI_2(c)\}$
5: $E(c) \leftarrow \{(v_1, v_2) | (v_1, v_2) \in E \wedge v_1, v_2 \in V(c)\}$
6: $\xi_L^{new} = \{\phi(e) | e \in E(c)\}$
7: $\eta_L^{new} = \{\eta(v) | v \in V(c)\}$
8: **for each** $(v_1, v_2) \in V(c) \times V_L^{node}(c)$ **do**
9: includeEdge$(L, (v_1, v_2), E_L^{new}, (v_1, v_2), \xi_L^{new})$
10: **end for**
11: **for each** $(v_1, v_2) \in E$ s.t. $\exists c_t \in V(c)$ s.t. $v_1 \in c_t$
12: and $v_2 \in V_L^{node}(c)$ **do**
13: includeEdge$(L, (v_1, v_2), E_L^{new}, (c_t, v_2), \xi_L^{new})$
14: **end for**
15: $V_L^{new} = (V_L \setminus \{c\}) \cup V(c)$
16: $E_L^{new} = (E_L \setminus \{(v, c) | (v, c) \in E_L\}) \cup E_L^{new} \cup E(c)$
17: $\xi_L^{new} = (\xi_L \setminus \{\xi_L((v, c)) | (v, c) \in E_L\}) \cup \xi_L^{new}$
18: $\eta_L^{new} = (\eta_L \setminus \{\eta_L(c)\}) \cup \eta_L^{new}$
19: **return** $(V_L^{new}, E_L^{new}, \eta_L^{new}, \xi_L^{new})$

activities carried out by the algorithm are the same one discussed for the $\texttt{zoom}_{\texttt{IN}}$ operation when dealing with leaf meta-nodes.

D. THE $\texttt{zoom}_{\texttt{CLASS}}$ OPERATION

When atomic-nodes of different clusters are reported in the current multi-resolution visualization, it might be useful to go back to a visualization in which only meta-nodes are reported. For this purpose, we introduce the $\texttt{zoom}_{\texttt{CLASS}}$ operation. Differently from all the other operations, this one is not applicable to a specific node of the current visualization but to the entire visualization. Its effect is to remove atomic-nodes and substitute them with the meta-node representing their most specific cluster. This operation corresponds to the application of $\texttt{zoom}_{\texttt{OUT}}$ operations of atomic-nodes.

Definition 10 ($\texttt{zoom}_{\texttt{CLASS}}$ Operation): Let G_L be a multi-resolution visualization of a graph G according to $ecc = ((c_1, \ldots, c_k), \sigma)$ and a cluster hierarchy C. The

Algorithm 6 $\texttt{zoom}_{\texttt{CLASS}}$

Input: the cluster tree $C = (V_C, E_C)$,
the graph $G = (V, E, \eta, \xi)$
the set L of addictive properties
the indexing structure $EI = (V_I, E_I, \phi_I)$
the slice (cc, σ)
the current visualization $G_L = (V_L, E_L, \eta_L, \xi_L)$
Output: a new visualization $G_L^{\texttt{new}} = (V_L^{\texttt{new}}, E_L^{\texttt{new}}, \eta_L^{\texttt{new}}, \xi_L^{\texttt{new}})$

1: $V_L^{\texttt{leaf}} \leftarrow \{c | c \in cc \wedge \sigma(c) = \texttt{true}\}$
2: $V_L^{\texttt{new}} \leftarrow V_L^{\texttt{meta}} \cup V_L^{\texttt{leaf}}$
3: $E_L^{\texttt{new}} = (E_L \setminus \{(v_1, v_2) \in E_L | \mathcal{F}(v_1) \in V_L^{\texttt{leaf}} \vee \mathcal{F}(v_2) \in V_L^{\texttt{leaf}}\})$
4: $\cup \{(c_1, c_2) \in E_I | c_1, c_2 \in cc\}$
5: $\xi_L^{\texttt{new}} = \{\phi_I((c_1, c_2)) | (c_1, c_2) \in E_L^{\texttt{new}}\}$
6: $\eta_L^{\texttt{new}} = \{\eta(v) | v \in V_L^{\texttt{new}}\}$
7: **return** $(V_L^{\texttt{new}}, E_L^{\texttt{new}}, \xi_L^{\texttt{new}}, \eta_L^{\texttt{new}})$

$\texttt{zoom}_{\texttt{CLASS}}$ operation allows to obtain a new expanded slice $ecc' = ((c_1, \ldots, c_k), \sigma')$, with $\sigma'(c) = \texttt{false}$ for $c \in \{c_1, .., c_k\}$. □

Example 10: Let $((B, D, E, G, I, J, K), \sigma)$ be an expanded slice where $\sigma(c) = \texttt{true}$ for all meta-nodes in the slice. This slice corresponds to the leaves of the cluster hierarchy and the visualization corresponds to the initial graph G in Figure 1. The application of $\texttt{zoom}_{\texttt{CLASS}}$ on it produces a visualization in which all the atomic-nodes are substituted by the corresponding leaf meta-nodes. □

Algorithm 6 reports the pseudo-code of the $\texttt{zoom}_{\texttt{CLASS}}$ operation and starts by identifying the meta-nodes $V_L^{\texttt{leaf}}$ that are expanded in the current visualization (line 1). The atomic-nodes belonging to such clusters need to be removed from the current visualization and substituted by the corresponding meta-nodes (line 2). Consequently, all the edges that involve the removed atomic-nodes need to be deleted from the new visualization. Edges contained in the `edgeIndex` involving vertices in $V_L^{\texttt{leaf}}$ should be included in the visualization (line 4). Finally, properties should be updated accordingly (lines 5,6).

E. CORRECTNESS AND COMPLEXITY RESULTS

The following theorems introduce correctness and complexity results on the application of our operators to a multi-resolution visualization G_L of a graph with respect to a cluster hierarchy C and the expanded slice ecc.

Theorem 12: The application of one of our operations to G_L always returns an expanded slice according to Definition 5. Moreover, the corresponding visualization algorithms are compliant with Definition 6. □

Theorem 13: For each pair of expanded slice ecc_1, ecc_2, there exists a sequence of operation $\texttt{zoom}_{\texttt{IN}}, \texttt{zoom}_{\texttt{OUT}}$ that allows to reach ecc_2 starting from ecc_1. □

Theorem 14: The complexity of our operations are:

$\texttt{zoom}_{\texttt{IN}}$	$\mathcal{O}(\lvert E\rvert) + \mathcal{O}(\lvert V\rvert)$
$\texttt{zoom}_{\texttt{OUT}}$	$\mathcal{O}(\lvert E_L\rvert) + max(\mathcal{O}(\lvert V\rvert), \mathcal{O}(\lvert V_C\rvert))$
$\texttt{zoom}_{\texttt{DEEP}}$	$\mathcal{O}(\lvert E\rvert) + \mathcal{O}(\lvert V\rvert)$
$\texttt{zoom}_{\texttt{CLASS}}$	$\mathcal{O}(\lvert E_L\rvert) + \mathcal{O}(\lvert V_L\rvert)$

□

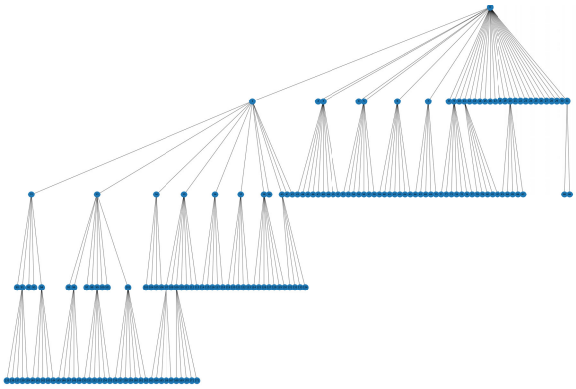


FIGURE 6. Example of hierarchy generated in one of our experiments.

V. EXPERIMENTAL RESULTS

Experiments were made to verify the effectiveness of the approach proposed for being used in a Web environment. The used machine is equipped with an i7 Intel core processor with 8 cores at 2.80 GHz, 120 Gbyte RAM, and a 1 Terabyte SSD. The machine runs an Ubuntu 12.04 Linux operating system.

All the algorithms described in this paper have been implemented in Python 3.7 and the original graph, the hierarchy, the *LI* and *EI* indexing structures, and the visualization graph have been stored in neo4j [14]. So, each time one of our operators is invoked, the visualization graph is updated by considering the hierarchy, the original graph (when needed), and the slice.

Many operations were simplified by exploiting the cypher query language with improvements in the overall execution times, like for example determining the nodes in the original graph that belong to a given cluster, or counting the cross clusters edges. However, it has also introduced some pitfalls. For example, neo4j does not allow to copy nodes/relations from one database to another and thus we were forced first to read them in main-memory and then write them back to the destination database. Another limitation is the lack of bulk operations for writing/updating a set of nodes/edges in a single request. This caused the creation of several requests to the server with negative effects on the execution times. Finally, injections by means of textual files are much faster than the use of the cypher create statement. So, in some cases, it is easier to extract inter-cluster relationships from a file, than using a cypher query for the same purpose. We point out these drawbacks because sometimes a great part of the execution times are due to the storing/updating of the visualization graph.

A. DATASETS

Two real datasets have been considered for evaluating the performances of our algorithms whose characteristics are reported in Table 2. The first one (denoted $G^{\texttt{sn}}$) contains unweighted graphs related to social networks and Amazon product networks made available by Stanford Large Network Dataset Collection (`https://snap.stanford.edu/`

TABLE 2. Description of real datasets and generated hierarchies.

(a) Social and amazon networks with corresponding hierarchies

G^{sn}	1	2	3	4	5	6	7	8	9	10
$\|V^{sn}\|$	762	4,039	3,490	265,214	3,455	4,369	196,591	1,088,092	410,236	168,114
$\|E^{sn}\|$	25,289	88,234	108,559	381,405	393,592	682,354	950,327	1,542,103	2,439,437	6,797,557
avg(Deg(G^{sn}))	66	44	62	3	228	312	10	3	12	81
max(Deg(G^{sn}))	387	1045	919	7636	1750	2185	14730	9	2760	35279
D(G^{sn})	0.088	0.011	0.018	1.0e-5	0.066	0.072	5.0e-5	3.0e-6	3.0e-5	4.8e-4
threshold	100	250	250	500	250	250	500	500	500	500
C^{sn}										
$\|V_C^{sn}\|$	24	49	54	1,092	44	63	1,420	9,976	3,152	1,254
leaves	19	39	42	941	33	49	1,278	9,689	2,895	1,069
depth	2	2	3	4	3	4	4	2	4	5
avg(|leaf|)	40	99	81	279	105	89	153	112	142	157
$\|EI^{sn}\|$	213	211	1,040	45,291	663	1,700	124,501	33,487	183,063	481,108

(b) Protein networks with corresponding hierarchies

G^{pro}	1	2	3	4	5	6	7	8	9	10
$\|V^{pro}\|$	1,457	11,996	20,408	18,785	6,441	13,535	6,467	20,380	14,740	19,336
$\|E^{pro}\|$	278,102	501,304	831,109	1,052,490	1,429,032	1,910,253	2,350,476	2,772,322	4,820,370	6,312,308
avg(Deg(G^{pro}))	382	84	81	112	444	282	727	272	654	653
max(Deg(G^{pro}))	1,112	673	386	1,274	2,438	2,377	3,640	1,747	4,176	4,396
D(G^{pro})	0.262	0.007	0.004	0.006	0.069	0.021	0.112	0.013	0.044	0.034
threshold	250	250	250	250	250	250	250	250	250	250
C^{pro}										
$\|V_C^{pro}\|$	14	174	272	260	103	173	88	236	213	255
leaves	10	145	224	230	92	148	79	211	194	227
depth	2	3	3	4	2	4	2	4	4	4
avg(|leaf|)	146	83	91	82	70	91	82	97	76	85
$\|EI^{pro}\|$	57	8,691	11,060	20,384	4,285	11,538	3,143	20,001	19,528	25,524

`data/`). The second one (denoted G^{pro}) are weighted networks of proteins extracted from different public sources and integrated through UNIPred-Web [12]. The reason for this choice is twofold and makes the two datasets valid test-beds for evaluating our approach: the diversity of the application contexts and their different complexities and topologies. For each graph, we report the number of nodes, the number of edges, the average and max degree of their nodes, and the density. Then, we generated the hierarchy by applying the hierarchical method proposed in [12] that relies on the use of the Louvain clustering algorithm [15] and applies a threshold on the size of the obtained clusters (when a cluster contains a number of nodes lesser than a given threshold, the cluster is not any longer split). Table 2 reports the adopted threshold for each graph. The characteristics of the obtained hierarchies are shown in Table 2(a) and Table 2(b). For each hierarchy, we report the number of nodes and leaf nodes, the hierarchy depth, the average number of nodes in the leaf clusters, and the number of edges in `EI`.

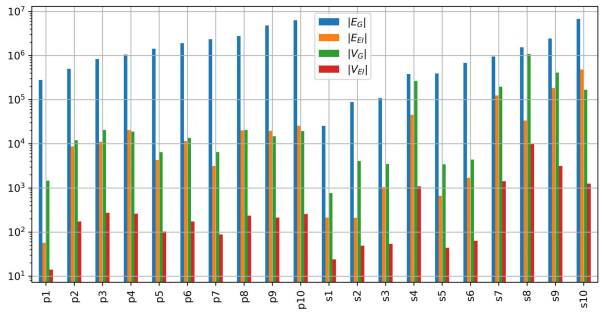
The considered graphs present a varying number of nodes and edges starting from small graphs (around 25,000 edges) to large graphs (around 6.8 million edges). The density of the considered graphs also varies between $3 * 10^{-6}$ and $2,6 * 10^{-1}$. The generated hierarchies arrive to have a depth of 5. Figure 6 shows an example of a hierarchy, with depth 4, generated for the graph G_5^{pro}. This graph presents more than 20 thousand nodes, but the hierarchy contains 272 meta-nodes leading to a better visual representation in a web page. Moreover, Figure 7(a) shows a comparison between the edges in G and those presented in EI, and the vertices of G and the vertices in EI. The diagram is represented according to a semi-logarithm scale in order to better interpret the result. Indeed, the vertices and edges in EI are always much lesser than those in G. Figure 7(b) shows another perspective of the same result by comparing the number of edges of `EdgeIndex` with respect to the edges of G: the size of `EdgeIndex` is several orders of magnitude lesser than the size of G. These diagrams prove the compactness of the obtained index. Next sections will show the suitability of the index in creating visualizations at different resolutions.

B. PERFORMANCE RESULTS

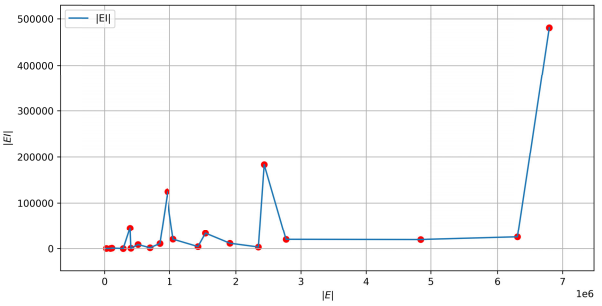
Several experiments have been done to assess the effectiveness of our solution. We have also checked whether the use of unweighted/weighted graphs has effects on the algorithms' performances, but there is no statistical evidence. The most significant experiments are reported.

1) CONSTRUCTION OF `LEAFINDEX` AND `EDGEINDEX`

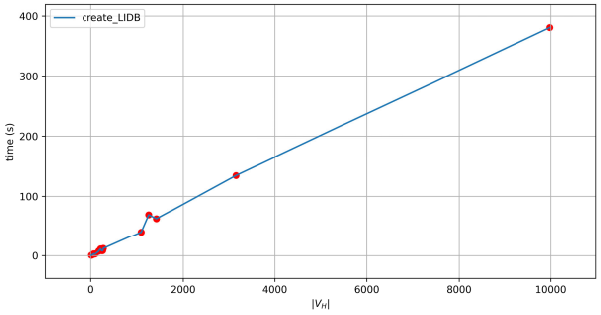
The first experiment is devoted to evaluate the time required for the construction of the two indexing structures. These operations are executed only once and offline. Figure 7(c) and Figure 7(d) report the execution times for the construction of `LeafIndex` (starting from an available hierarchy) and `EdgeIndex` comprehensive of the time required for storing the indices in the neo4j database. The time required for constructing the `LeafIndex` is often negligible also when the size of the graphs grow. This is due to the linear complexity of the developed algorithm. Things are different



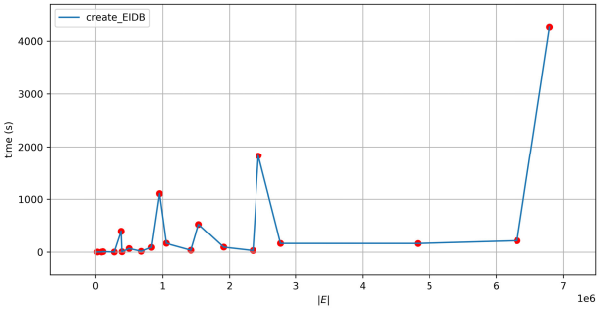
(a) Graphs vs `edgeIndex` edges and nodes



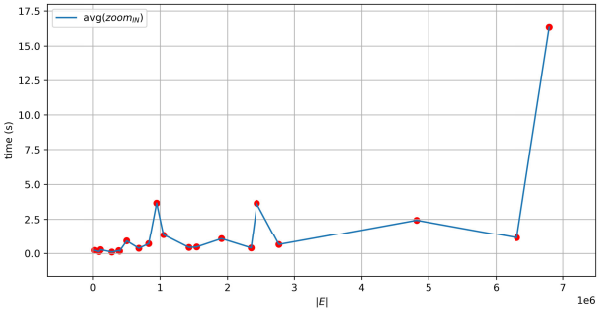
(b) Size of `edgeIndex` vs number of edges



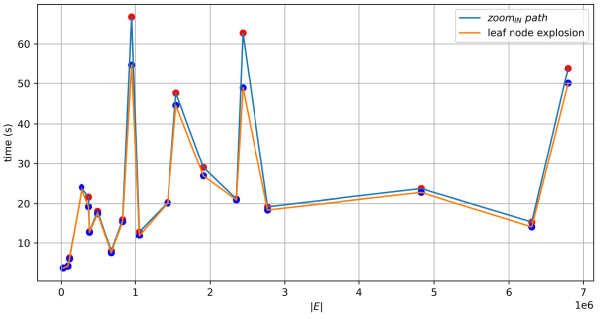
(c) `leafIndex`: creation and storage time



(d) `edgeIndex`: creation and storage time



(e) $\texttt{zoom}_{\texttt{IN}}$: avg time to unfold internal nodes



(f) $\texttt{zoom}_{\texttt{IN}}$: unfond leaf node vs entire path

FIGURE 7. **Experiments of main operations on real datasets.**

for the construction of `EdgeIndex`. Indeed, in this case, all the inter-meta-nodes relationships should be computed and their number is quadratic w.r.t. the number of nodes of the hierarchy. Therefore, `EdgeIndex` construction time is generally higher than `LeafIndex` construction time.

2) $\texttt{ZOOM}_{\texttt{IN}}$ AND $\texttt{ZOOM}_{\texttt{OUT}}$ OPERATIONS

The second experiment is devoted to the analysis of the cost of the $\texttt{zoom}_{\texttt{IN}}$ and $\texttt{zoom}_{\texttt{OUT}}$ operations in dealing with the internal nodes of the hierarchy. Through this kind of experiment we can evaluate the effectiveness of our indexing structures because the original graph does not need to be considered. Figure 7(e) reports the average time for the invocation of the $\texttt{zoom}_{\texttt{IN}}$ operation on all the internal nodes of the hierarchy starting from the root for our two datasets. A linear dependency can be identified on the number of edges of the original graph. The average time is mainly negligible (less than 5 sec.) with the exception of the graph with 6,8 million edges where the time moves to 17.5 sec. Starting from the visualization obtained by exploding all internal meta-nodes, the $\texttt{zoom}_{\texttt{OUT}}$ operation is invoked in reverse order until the visualization contains only the hierarchy root. The average time for each invocation is linear w.r.t. the number of edges of the graph. The execution time is mainly negligible (usually less than 0.5 sec) with a pick of 10 sec for the biggest graph.

3) NAVIGATION FROM THE ROOT TO THE LEAVES OF THE HIERARCHY

In order to simulate a possible graph exploration that a user might be interested in, we have created a max length path from the hierarchy root to the leaf-meta-node with the highest number of atomic-nodes. Therefore, in this case, we have to explode the higher number of meta-nodes and then introduce

in the visualization a high number of atomic-nodes. Then, we have taken the average time required to $\texttt{zoom}_{\texttt{IN}}$ all the meta-nodes in the path. In this case, the execution time is higher than in the previous cases. The reason is that the explosion of the leaf hierarchy node requires querying the original graph to extract the corresponding atomic-nodes. This behavior is also confirmed by Figure 7(f) where two lines are proposed: one represents the execution time of all the $\texttt{zoom}_{\texttt{IN}}$ path steps, and the other represents only the execution time of the last step. As the reader can see, the execution of the internal steps is negligible w.r.t. the explosion of the leaf node and this is due to the presence of the `EdgeIndex` that for internal nodes of the hierarchy does not require any complex computation. We have also considered the execution of the $\texttt{zoom}_{\texttt{OUT}}$ operation on the same path but in reverse order. Usually, the average execution time is less than 0.24 sec (and in the worst case is less than 0.5 sec).

VI. RELATED WORK

Many approaches were proposed for the visualization and interactive analysis of complex graphs [1] and for the design of visualization and navigational tools [7], [8].

A. HIERARCHICAL CLUSTERING APPROACHES

Different approaches have been proposed for the identification of communities/clusters/groups in complex networks that share commonalities according to a distance function. Although no formal definition of cluster is universally accepted [16], a largely adopted measure to quantify the quality of clusters is the minimization of *modularity function* [17] where nodes in the single clusters are more likely to be connected than expected in a random network null model [18]. Since the global optimization of the modularity is a well known NP-hard problem [19], different local heuristics are used [20], [21], [22], [23], [24], [25], [26].

When the clustering is hierarchical, the structure imposed by the hierarchy can be exploited both for the visualization of the clusters and for supporting navigation operations through the clusters [1]. These facilities allow the visualization of the network at different granularities. At coarser granularities, the general organization of the network can be inspected, whereas at finer granularities, it is possible to appreciate peculiarities of specific parts of network [6]. The computation of the hierarchy can be realized by means of agglomerative [20], [27] and divisive algorithms [22]. More advanced techniques rely on the construction of a multi-layer network, where each layer has a dedicated scale parameter [28]. Usually, because of their good scaling property, agglomerative methods are preferred for large-sized networks and when used in web-tools (that should provide fast responses).

Approaches and systems for the visual exploration of the hierarchical communities have been proposed [29], [30] as well as multi-resolution visualizations of cellular network processes [31] and biological pathways [32]. However, their focus is mainly on the rendering of the meta-nodes on the canvas, whereas our focus is on the data and indexing structures adopted for easily retrieving and preparing big graphs to be rendered at different granularity levels.

B. ABSTRACT VISUAL REPRESENTATIONS

The indexing structures that we have presented in this paper can be exploited in graph visualization tools (like PivotGraph [11], Zame [10], and Motif Simplifications [9]) to increase the scalability of the visualization procedures and to summarise nodes and edges with shared characteristics and thus produce succinct visualizations of important properties of large networks. In this case, instead of applying clustering techniques, nodes are classified according to a property (e.g. a categorical attribute, geographic region, topological feature) and a hierarchy can be generated on it. By using this type of aggregation, as suggested in [2], interesting relationships between attribute, properties, and topological features can be revealed. However, to be interactively exploited, aggregations need to be achieved in a preprocessing step, as our indexing structures.

C. WEB-BASED GRAPH VISUALIZATION TOOLS

Several tools have been proposed in the last few year for the visualization and interaction of graph-based networks, especially in the context of protein networks (e.g. Genemania [3], ZoomOut [33], PINV [34], STRING [4], IMP 2.0 [35], and UNIPredWeb [12], [36]) for the interactive exploration of protein networks (a comprehensive survey can be found in [37]). Few of them supports the possibility to generate clusters of nodes and exploit them for the navigation of the network topology (i.e. STRING [4], UNIPredWeb [12], [36]). UNIPredWeb shows communities of nodes at different levels of detail and can expand or collapse community on-demand. On the contrary, STRING can just cluster nodes by coloring them, built without allowing users to expand or collapse the displayed communities. ZoomOut can apply clustering methods using a set of computed descriptors for each network and all networks can be visualized as single nodes of a super-network, were interconnections among networks are based on the calculated clustering distances. However, this tool does not identify communities/clusters inside each network but only clusters of networks. In this paper, our focus is on the back-end of the multi-resolution visualization when meta-nodes are used at different granularities by providing properties of the data structures and algorithms.

VII. CONCLUSION AND FUTURE WORK

In this paper we have proposed the use of a cluster hierarchy for the navigation of big property graphs at different resolution levels. Cluster of atomic-nodes are substituted with meta-nodes and different kinds of edges are introduced in the visualization to show the relationships among meta-nodes and atomic-nodes. Different indexing structures and the concept of slice have been introduced to make faster the folding and unfolding operations that can be applied on the visualization. An incremental approach

has been adopted in the implementation of the operations in order to minimize the part of the visualization affected by the application of the operations. The paper also contains an in depth analysis of the properties of the proposed data structures and the time complexity of our operations.

The proposed approach has been applied to real graphs of different sizes. Graphs with millions of edges can be handled in a reasonable amount of time (usually less than 10 sec with a pick of 20 sec for graphs with almost 6 million edges). The obtained hierarchy usually counts an average number of 200 nodes with picks of 10 thousand nodes for sparse graphs with a low density. We remark that visualizations with such a number of nodes can be done in a web application. In the implementation, we used neo4j for storing and updating the graph visualization. However, a good part of the time (around 90%) is devoted to the neo4j reading and writing operations. We think that execution times can be improved in new releases of neo4j where these operations will be optimized.

To the best of our knowledge, we are not aware of other approaches similar to ours and this is the reason for the lack of comparison experiments. We are however working on ablation studies to demonstrate the effectiveness of our indexing structures and algorithms.

As future work we would like to combine the back-side services described in this paper with new visual artifacts that will improve the multi-resolution visualization of the property graphs. Moreover, we wish to extend the clustering algorithm used in this paper to provide a better organization of the clusters from a visualization point of view. Indeed, sometimes the produced clusters might be merged to reduce their number by taking into account the dimension of the visualization canvas. This kind of aggregation does not depend on the similarity measure used for clustering together the nodes but from external parameters (size of the canvas, density or level of connectiveness of the graph). Finally, we would like to extend the methodology for dealing with temporal graphs thus being able to consider also the time in the generation of the hierarchy and in the visualization at different granularities.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their useful suggestions for improving the quality of their work and the members of Anacleto Laboratory (https://anacletolab.di.unimi.it/) with the University of Milano for useful discussions on the topic of this article.

REFERENCES

[1] I. Herman, G. Melancon, and M. S. Marshall, "Graph visualization and navigation in information visualization: A survey," *IEEE Trans. Vis. Comput. Graphics*, vol. 6, no. 1, pp. 24–43, Jan./Mar. 2000.

[2] C. Nobre, M. Meyer, M. Streit, and A. Lex, "The state of the art in visualizing multivariate networks," *Comput. Graph. Forum*, vol. 38, no. 3, pp. 807–832, Jun. 2019.

[3] M. Franz, H. Rodriguez, C. Lopes, K. Zuberi, J. Montojo, G. D. Bader, and Q. Morris, "GeneMANIA update 2018," *Nucleic Acids Res.*, vol. 46, no. W1, pp. W60–W64, Jul. 2018.

[4] D. Szklarczyk, A. L. Gable, D. Lyon, A. Junge, S. Wyder, J. Huerta-Cepas, M. Simonovic, N. T. Doncheva, J. H. Morris, P. Bork, L. J. Jensen, and C. V. Mering, "STRING v11: Protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets," *Nucleic Acids Res.*, vol. 47, no. D1, pp. D607–D613, Jan. 2019.

[5] N. T. Doncheva, J. H. Morris, J. Gorodkin, and L. J. Jensen, "Cytoscape StringApp: Network analysis and visualization of proteomics data," *J. Proteome Res.*, vol. 18, no. 2, pp. 623–632, Feb. 2019.

[6] N. Elmqvist and J.-D. Fekete, "Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines," *IEEE Trans. Vis. Comput. Graphics*, vol. 16, no. 3, pp. 439–454, May 2010.

[7] T. Munzner, "A nested model for visualization design and validation," *IEEE Trans. Vis. Comput. Graphics*, vol. 15, no. 6, pp. 921–928, Nov. 2009.

[8] M. Sedlmair, M. Meyer, and T. Munzner, "Design study methodology: Reflections from the trenches and the stacks," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 12, pp. 2431–2440, Dec. 2012.

[9] C. Dunne and B. Shneiderman, "Motif simplification: Improving network visualization readability with fan, connector, and clique glyphs," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, Apr. 2013, pp. 3247–3256.

[10] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J.-D. Fekete, "ZAME: Interactive large-scale graph visualization," in *Proc. IEEE Pacific Vis. Symp.*, Mar. 2008, pp. 215–222.

[11] M. Wattenberg, "Visual exploration of multivariate graphs," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, New York, NY, USA, Apr. 2006, pp. 811–819.

[12] P. Perlasca, M. Frasca, C. T. Ba, J. Gliozzo, M. Notaro, M. Pennacchioni, G. Valentini, and M. Mesiti, "Multi-resolution visualization and analysis of biomolecular networks through hierarchical community detection and web-based graphical tools," *PLoS ONE*, vol. 15, no. 12, pp. 1–28, Dec. 2020.

[13] M. Golfarelli and S. Rizzi, *Data Warehouse Design: Modern Principles and Methodologies*. New York, NY, USA: McGraw-Hill, 2009.

[14] A. Vukotic, N. Watt, T. Abedrabbo, D. Fox, and J. Partner, *Neo4j in Action*, vol. 22. New York, NY, USA: Manning Shelter Island, 2015.

[15] V. A. Traag, "Faster unfolding of communities: Speeding up the Louvain algorithm," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 92, no. 3, Sep. 2015, Art. no. 032801.

[16] S. Fortunato, "Community detection in graphs," *Phys. Rep.*, vol. 486, nos. 3–5, pp. 75–174, Feb. 2010.

[17] M. E. J. Newman, "Modularity and community structure in networks," *Proc. Nat. Acad. Sci. USA*, vol. 103, no. 23, pp. 8577–8582, Jun. 2006.

[18] M. Newman, *Networks: An Introduction*. London, U.K.: Oxford Univ. Press, 2010.

[19] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner, "On modularity clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 2, pp. 172–188, Feb. 2008.

[20] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech., Theory Exp.*, vol. 2008, no. 10, Oct. 2008, Art. no. P10008.

[21] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970.

[22] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proc. Nat. Acad. Sci. USA*, vol. 99, no. 12, pp. 7821–7826, Jun. 2002.

[23] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, Jun. 2005.

[24] P. Pons and M. Latapy, "Computing communities in large networks using random walks," *J. Graph Algorithms Appl.*, vol. 10, no. 2, pp. 191–218, 2006.

[25] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 70, no. 6, Dec. 2004, Art. no. 066111.

[26] A. Noack and R. Rotta, "Multi-level algorithms for modularity clustering," in *Proc. Int. Symp. Exp. Algorithms*, in Lecture Notes in Computer Science, vol. 5526, Dec. 2009, pp. 257–268.

[27] A. Clauset, C. Moore, and M. Newman, "Structural inference of hierarchies in networks," in *Statistical Network Analysis: Models, Issues, and New Directions*, vol. 4503. Pittsburgh, PA, USA: Springer, 2007, pp. 1–13.

[28] A. Ashourvan, Q. K. Telesford, T. Verstynen, J. M. Vettel, and D. S. Bassett, "Multi-scale detection of hierarchical community architecture in structural and functional brain networks," *PLoS ONE*, vol. 14, no. 5, pp. 1–36, May 2019.
[29] D. Auber and F. Jourdan, "Interactive refinement of multi-scale network clusterings," in *Proc. Int. Conf. Inf. Vis.*, 2005, pp. 703–709.
[30] B. Renoust, G. Melançon, and T. Munzner, "Detangler: Visual analytics for multiplex networks," *Comput. Graph. Forum*, vol. 34, no. 3, pp. 321–330, Jun. 2015.
[31] O. O. Ortega and C. F. Lopez, "Interactive multiresolution visualization of cellular network processes," *iScience*, vol. 23, no. 1, Jan. 2020, Art. no. 100748.
[32] F. Paduano and A. Forbes, "Extended LineSets: A visualization technique for the interactive inspection of biological pathways," in *Proc. Symp. Biol. Data*, 2015, p. S4.
[33] E. I. Athanasiadis, M. M. Bourdakou, and G. M. Spyrou, "ZoomOut: Analyzing multiple networks as single nodes," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 12, no. 5, pp. 1213–1216, Sep. 2015.
[34] G. A. Salazar, A. Meintjes, G. K. Mazandu, H. A. Rapanoël, R. O. Akinola, and N. J. Mulder, "A web-based protein interaction network visualizer," *BMC Bioinf.*, vol. 15, no. 1, p. 129, Dec. 2014.
[35] A. K. Wong, A. Krishnan, V. Yao, A. Tadych, and O. G. Troyanskaya, "IMP 2.0: A multi-species functional genomics portal for integration, visualization and prediction of protein functions and networks," *Nucleic Acids Res.*, vol. 43, no. W1, pp. W128–W133, Jul. 2015.
[36] P. Perlasca, M. Frasca, C. T. Ba, M. Notaro, A. Petrini, E. Casiraghi, G. Grossi, J. Gliozzo, G. Valentini, and M. Mesiti, "UNIPred-web: A web tool for the integration and visualization of biomolecular networks for protein function prediction," *BMC Bioinf.*, vol. 20, no. 1, p. 422, Dec. 2019.
[37] G. A. Pavlopoulos, A.-L. Wegener, and R. Schneider, "A survey of visualization tools for biological network analysis," *BioData Mining*, vol. 1, no. 1, p. 12, Dec. 2008.

MARCO MESITI received the master's and Ph.D. degrees in computer science from the University of Genova, in 1998 and 2003, respectively. He is currently an Associate Professor with the Department of Computer Science Giovanni degli Antoni, Università degli Studi di Milano. He was a Visiting Researcher with Bellcore (then Telcordia Technologies), Morristown (New Jersey), in different periods, from 1998 to 2000. His research interests include the integration, querying, and visualization of different kinds of information (structured and semi-structured) according to different data models (relational, graph, and nosql). Moreover, he has been involved in different projects for graph data integration, node and edge prediction, and heterogeneous network visualization. On these topics, he has published more than 100 papers in international conferences and journals. He is an Associate Editor of the *Data Science and Engineering* journal (Springer) and *Applied Sciences* (MDPI).

MARIO PENNACCHIONI received the master's degree in mathematics. He has been a Research Associate with the Department of Computer Science Giovanni degli Antoni, Università degli Studi di Milano, since 2017. Previously, he has worked either as a computer scientist free lance or an employee in several companies (e.g., banks, marketing analysis). His area of expertise is mainly in data management and data mining.

PAOLO PERLASCA received the master's and Ph.D. degrees in computer science from the University of Milano, in 1999 and 2003, respectively. He is currently an Assistant Professor with the Department of Computer Science Giovanni degli Antoni, Università degli Studi di Milano. He has worked in the area of data security, integrity and protection, and in the management and protection of intellectual property and digital rights. His current research interests include the development of efficient systems for the analysis, merging, and the visualization of large heterogeneous networks.

Open Access funding provided by 'Università degli Studi di Milano' within the CRUI CARE Agreement