IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 16 October 2022, accepted 28 October 2022, date of publication 1 November 2022, date of current version 8 November 2022.
Digital Object Identifier 10.1109/ACCESS.2022.3218884

RESEARCH ARTICLE

BrowVis: Visualizing Large Graphs in the Browser

LUCA CONSALVI, WALTER DIDIMO, GIUSEPPE LIOTTA, (Senior Member, IEEE), AND FABRIZIO MONTECCHIANI
Dipartimento di Ingegneria, Università degli Studi di Perugia, 06125 Perugia, Italy
Corresponding author: Fabrizio Montecchiani (fabrizio.montecchiani@unipg.it)

This work was supported in part by MIUR through "AHeAD: efficient Algorithms for HArnessing networked Data" under Grant 20174LF3T8; and in part by the University of Perugia, Fondi di Ricerca di Ateneo 2021, Project "AIDMIX—Artificial Intelligence for Decision Making: Methods for Interpretability and eXplainability."

ABSTRACT A recent stream of research focuses on building high-performance data analysis and management systems that run completely in the browser. Indeed, today personal devices offer non-trivial amount of computing power, while the latest Web browsers provide powerful JavaScript engines. On the other hand, the use of visualization to present and analyze networks is taking a leading role in conveying information and knowledge to users that operate in multiple domains. In this scenario, the aim of our research is to explore the scalability limits of a system that executes the full graph visualization pipeline entirely in the browser. In this paper, we present BrowVis, a self-contained system to compute interactive visualizations of large graphs in the browser. Experiments show that, on a common laptop, BrowVis can visualize graphs with thousands of elements in seconds, as well as graphs with millions of elements in minutes. Once the initial visualization has been computed, BrowVis makes it possible to interactively explore the represented graph by following a details-on-demand paradigm. The use of BrowVis in practice is demonstrated by a case study dealing with a real-world scientific collaboration network.

INDEX TERMS Network visualization, visual analytics, in-browser computing.

I. INTRODUCTION

Graphs appear as a natural model for representing data in various fields and application domains, such as for instance artificial intelligence [1], finance [2], recommender systems [3], social network analysis [4], and tourism [5]. In particular, the use of visualization to present and analyze networked data is taking a leading role in conveying information and knowledge to users that operate in the above mentioned domains (see, e.g., [6]). Indeed, when dealing with large graphs, a recent extended survey [7] revealed that visualization is a very popular and central task in graph processing pipelines. On the other hand, designing algorithms to produce valuable visualizations of large graphs is difficult, and it is reported in [7] as one of the most pressing graph processing challenges.

The problem of producing graph visualizations can be decomposed into a two-step graph processing pipeline, with iterations that might be triggered by user interaction. In the first step, a layout of the input graph is computed, which involves the assignment of geometric representations to the vertices and to the edges of the graph. In the second step, the layout is rendered on the screen through a user interface, which often enables the exploration of the displayed data via interaction primitives. The layout step involves the design of efficient algorithms to optimize some aesthetic criteria while satisfying given conventions and constraints. For instance, in the popular node-link paradigm, vertices are represented as points, edges are straight-line segments, and the layout should both avoid the clutter given by overlapping features and highlight possible symmetries in the underlying graph. The vast majority of the algorithms used to produce node-link layouts of large graphs follows force-directed methods [8], [9], [10], which usually yield super-linear time complexities. In fact, for the sake of efficiency, layout algorithms can be run remotely on powerful servers or cloud computing infrastructures (see, e.g., [11] and [12]). The rendering step requires a careful use of the graphics system on the client side to avoid inefficiencies and artifacts, as well as the design of suitable

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Da Lin.

interaction paradigms to support an effective exploration of the conveyed data. It is worth mentioning that interactive rendering paradigms may involve the use of visual abstractions of the input layout, aimed to reduce both the overall clutter of the visualization and the computational cost of displaying a huge amount of geometric elements (see, e.g., [13]).

Today personal devices offer non-trivial amount of computing power and Web browsers provide powerful JavaScript engines. As a consequence, a recent stream of work focuses on building high-performance data analysis and management systems that run completely in the browser. A limited list of examples include the following: El Gebaly and Lin [14] present an analytical relational DBMS implemented in JavaScript that runs within the browser; Lin [15] describes a self-contained JavaScript-based search engine; Lee et al. [16] propose a JavaScript implementation of a keyword spotting system that can be deployed directly on user devices.

In this paper, we aim at exploring the scalability limits of a system that executes the entire graph visualization pipeline relying only on the JavaScript processing engine of the browser. The main motivation of our work is twofold: on the one hand, having the whole visualization produced in the browser cuts off network latency, enables offline usage, avoids security and privacy issues related to the transit and remote storage of the data, and removes any external dependency. On the other hand, while modern web technologies (and in particular JavaScript) can be used to effectively tackle large graph datasets [17], existing graph visualization tools struggle to achieve satisfactory performance already with graphs having up to few hundred thousand elements [18].

Our main contribution is as follows:

- We describe BrowVis, a self-contained system to compute interactive visualizations of large graphs in the browser. BrowVis significantly differs from existing Web technologies as it combines two best-in-class techniques to carry out the entire graph processing pipeline. Namely, a layout of the input graph is computed with a JavaScript porting of FM^3 [19], [20], a multi-level force-directed algorithm originally developed in C++. To perform rendering and interaction, BrowVis incorporates and adapts the main ideas behind LaGO [13], an OpenGL-based implementation of a technique to interactively render massive node-link layouts with adjustable level of abstraction.
- We provide a publicly available proof-of-concept implementation of BrowVis, and we report the outcome of an extensive experimental analysis aimed at assessing its performance. The experiments show that, on a common laptop, BrowVis can visualize graphs with several thousand elements in seconds, as well as graphs with millions of elements in minutes. Moreover, once the initial visualization has been computed, BrowVis makes it possible to interactively explore the represented graph by following a details-on-demand paradigm.
- We finally showcase the features of BrowVis's user interface with a case study dealing with a real-world scientific collaboration network. BrowVis quickly produces a first visualization of the network, and it then supports a smooth exploration of the drawing. The interaction reveals, in particular, the presence of communities of authors, as well as the role of some central scientists that bridge different communities.

The remainder of this paper is organized as follows. Section II contains the necessary background on graph visualization, along with a brief overview of the related literature. Section III provides a detailed description of BrowVis and of the rationale behind each of its design choices. Section IV describes the experimental analysis and its outcome. Section V reports on the case study. Section VI concludes the paper with a short discussion and future research directions.

II. BACKGROUND AND RELATED WORK

In this section we provide some background on the two key ingredients of our system, namely force-directed layout algorithms and rendering techniques, along with a brief overview of the related literature.

A. FORCE-DIRECTED LAYOUT ALGORITHMS

Force-directed algorithms are the most common solution to the problem of computing node-link layouts of general unrestricted graphs. They follow two basic principles: (*i*) edges should not be too long and hence adjacent vertices should be drawn near to each other; (*ii*) vertices should not overlap and should evenly distribute on the drawing area. These two principles can be encoded in a system of forces acting on the vertices of the input graph. We point the reader to the extensive surveys of Cheong and Si [8] and by Kobourov [10] on the vast literature on force-directed algorithms, as well as to the surveys by Hu and Shi [9] and by Landesberger et al. [21] that are focused on the visualization of large graphs. We also remark that the versatility of force-directed algorithms makes them suitable to visualize dynamic graphs [22], as well as clustered and compound graphs [23], [24]. Common to all force-directed algorithms are a model of the system of forces acting on the vertices and an iterative algorithm to find a static equilibrium of this system, which represents the final layout of the graph. In terms of running time, the main bottleneck lies in the fact that each vertex interacts with all other vertices, giving rise to an overall quadratic number of forces in each iteration of the algorithm. To alleviate this problem, different authors proposed spatial decomposition techniques to approximate forces acting between vertices that are far from each other [25], [26]. A quantum leap towards the applicability of force-directed algorithms to larger graphs is represented by *multilevel force-directed algorithms*, introduced in [26], [27], and [28]. Algorithms in this family proceed along a framework that roughly works as follows: first the input graph is iteratively simplified via coarsening techniques, giving rise to a stack of coarser graphs; second, such a stack is traversed backward and a final layout of the original graph is obtained by progressively computing a layout for each intermediate graph in the sequence. As experimentally observed, FM^3 [29], [30] is one of the most effective multilevel force-directed algorithms, as it produces less edge crossings and fewer vertex overlaps [20], [31].

In order to unleash the power of modern computing infrastructures, different implementation choices have been investigated; we briefly describe a restricted list of examples. The first attempts to scale force-directed algorithms to very large graphs exploit the power of GPUs [32], [33], [34], [35]. They can draw graphs with a few million edges, but their development requires a low-level implementation tied to the computing platform. Parallel and distributed approaches have also been considered. Meyerhenke et al. [36] present a C++ implementation based on OpenMP of a layout algorithm using the maxent-stress metric for the layout optimization. Mueller et al. [37] and Chae et al. [38] propose force-directed algorithms that use multiple large displays; vertices are evenly distributed on the different displays, each associated with a different processor, which is responsible for computing the positions of its vertices. Tikhonova and Ma [39] present a parallel force-directed algorithm that can run on graphs with few hundred thousand edges and experimented it on the BigBen Cray XT3 cluster. More recently, a series of works has pursued the use of modern Big Data frameworks. Hinge and Auber [40] describe a distributed force-directed algorithm implemented in Apache Spark (using the GraphX library). Hinge et al. [41] present a multilevel force-directed algorithm, also implemented in Apache Spark. Arleo et al. [11], [12] describe both force-directed and multilevel force-directed algorithms based on the Apache Giraph platform. Their multilevel implementation follows the ideas of FM^3 and can draw graphs with millions of edges in few minutes.

Finally, sampling and sparsification approaches have been very recently proposed to obtain sublinear force computation schemes [42].

B. RENDERING TECHNIQUES

The goal of rendering techniques is to avoid clutter and over-plotting, which are undesirable effects both in terms of readability and efficiency. Clutter occurs when many vertices and edges of the graph are drawn in small portions of the screen, giving rise to ambiguous blobs of pixels. This issue is unavoidable if we insist on drawing each single vertex and edge of a large graph containing more elements than the pixels the screen can offer. Moreover, dense portions of the graph force many edges to traverse common areas of the screen which, in turn, gives rise to plotting over the same pixels multiple times, a severe problem in terms of efficiency. Inspired by Shneiderman's mantra [43] "Overview first, zoom and filter, then details-on-demand", several authors proposed multilevel visualizations aimed at computing multiple abstractions of the input graph (see, e.g., [26], [44], [45], and [46]). While seminal approaches in this direction bundle together layout and rendering, Zinsmaier et al. [13] propose an interactive rendering technique with adjustable levels of detail that operates directly on a given layout and does not require precomputed hierarchies or meshes. The approach in [13] consists of a combination of edge accumulation with density-based vertex aggregation, and its implementation exploits graphics hardware for speeding-up the computation. In the same spirit, Perrot and Auber [47] describe a multilevel system that works for any given layout in input. The main differences with respect to [13] are the use of different algorithms for vertex and edge aggregation and an implementation based on distributed platforms for Big Data processing.

While all above papers provide fundamental scientific groundwork for our research, none of the above techniques is conceived to run entirely in the browser. On the other hand, there exist many JavaScript-based libraries that can deal with both the layout and the rendering steps. In particular, Han et al. [18] present NetV.js, a JavaScript library for the visualization of large graphs, and compare its performance with several other JavaScript libraries for graph visualization, namely Cytoscape.js [48], D3.js [49], Sigma.js [50], and Stardust.js [51]. Based on their experiments, the authors conclude that Stardust.js and D3.js can render up to a total of one hundred thousand elements (both vertices and edges), while NetV.js can render up to a total of one million elements showing at least one frame per second. The main drawback of the experimental analysis in [18] is that no performance metric is reported (e.g., runtime or memory footprint) other than the framerate. Moreover, and most importantly, none of the above libraries provide abstractions but instead draw each single vertex and edge of the graph, thus incurring into both clutter and over-plotting.

In addition to the general-purpose libraries mentioned above, we briefly discuss some related systems for processing networks entirely in the browser. Carbonic [52] is a Javascript-based system for interactive visual exploration and editing of compound graphs. The main similarities between Carbonic and BrowVis are the adoption of a multiscale visualization paradigm and of edge bundling techniques to cope with visual clutter. On the other hand, the main goal of Carbonic is to effectively support the visualization and exploration of the hierarchical structure underlying compound graphs, in particular, it employes sunburst diagrams rather than node-link diagrams. NDExEdit [53] is an in-browser application to support edits and enhancements of biological networks. NDExEdit is compatible with Cytoscape and NDEx and fits the needs of collaborative workflows in the biological domain.

III. THE BrowVis SYSTEM

In this section we describe the design of BrowVis, a self-contained system to compute interactive visualizations of large graphs fully in the browser. Our system embraces the concept of multilevel visualization in order to achieve both readability and scalability. The graph processing pipeline

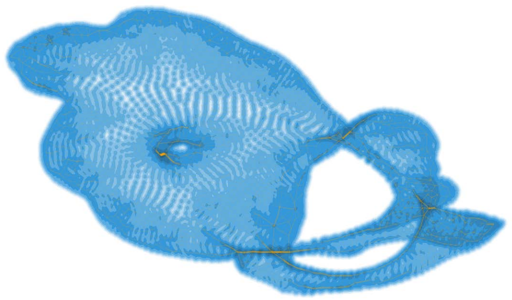

FIGURE 1. A graph with a regular structure and that contains about 20 000 elements (vertices and edges) visualized with BrowVis.

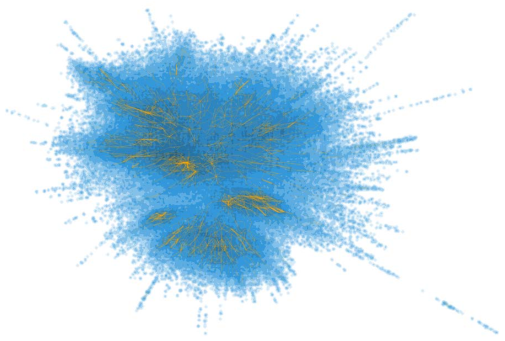

FIGURE 2. A complex network with about 1 200 000 elements visualized with BrowVis.

adopted by BrowVis is described below. The source code is publicly available.[1] Figs. 1 and 2 illustrate two graphs with different structures and sizes visualized with BrowVis.

A. LAYOUT

As discussed in Section II, there exists a vast literature concerning force-directed algorithms, and the design of an original layout algorithm is beyond the scope of this paper. Instead, our choice is to rely on the OGDF implementation of FM^3 [19], [20], a robust implementation already experimented in multiple works.

We ported the C++ implementation of FM^3 into WebAssembly,[2] an open standard that defines a portable binary-code format for executable programs and that provides a JavaScript API. In particular, we used Emscripten,[3] an open source software to compile C and C++ code into WebAssembly; the output code is compact and runs at near-native speed. To avoid blocking the user interface during the layout computation, we use a dedicated Web worker that runs in background.

As it will be further clarified in the remainder of this section, the layout step of the pipeline is executed only once in BrowVis, whereas the rendering step may be repeated multiple times depending on user interaction.

[1]https://github.com/Luk4e/graph_visualization
[2]https://webassembly.org/
[3]https://emscripten.org/

FIGURE 3. Color palette used for the density field.

B. RENDERING

To perform the rendering, BrowVis engineers the main ideas behind LaGO [13]. The original implementation of LaGO exploits the OpenGL rendering pipeline, which is not conceived for Web browsers. Instead, BrowVis exploits the WebGL technology, which is based on OpenGL ES, a subset of OpenGL. Specifically, we utilize pixi.js,[4] a general-purpose library that offers low-level primitives for 2D rendering. Moreover, the computed layout is rendered by means of a dedicated Web worker that runs in background.

At high-level, BrowVis first accumulates vertices based on density fields and it then exploits the obtained fields to aggregate edges. To accumulate vertices, the system adopts a kernel density estimation (KDE) with Gaussian kernels (hence following the approach in [13]). Formally, let $G = (V, E)$ be a graph with n vertices and m edges and let Γ be a drawing of G in output from the layout step. For each vertex $v_i \in V$ $(i \in \{1, \dots, n\})$, let $p_i = (x_i, y_i)$ be the point representing v_i in Γ. For each pixel $p = (x, y)$ of our drawing area, the density field function at p is defined as follows and depends on the parameter σ:

$$D_f(x, y, \sigma) = \sum_{i=1}^{n} \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}(x-x_i+y-y_i)^2}$$

Based on the zoom level, part of the drawing Γ may be outside the drawing area; in such a case, the density field is computed only with respect to the vertices that are part of the visible area, plus those vertices that lie in a frame of fixed size around it. Once a density value has been computed for each pixel, the values are normalized in the range [0, 1] and discretizes by using a constant number of levels mapped to the color palette illustrated in Fig. 3. To speed-up our implementation, rather than applying the above formula for each pixel and for each vertex, we generate a single density field prototype and move it iteratively on top of each vertex in the drawing area, in order to update only the pixels in a neighborhood of that vertex.

To aggregate edges, the idea is to identify clusters in the density field and only represent inter-cluster edges. We use a hill climbing algorithm to move each endpoint of an edge to the highest point around it, called *peak* in the following. After this operation, there will be bundles of aggregated edges (those whose endpoints are mapped to the same peaks). In particular, inner-cluster edges (those whose endpoints are both mapped to the same peak) disappear, while inter-cluster edges are emphasized. Let the *weight* of an inter-cluster edge be the number of original edges aggregated into this edge. Similarly as for the density field, the weights are normalized and discretized by using a constant number of levels mapped to the color opacity and to the line thickness

[4]https://pixijs.com/

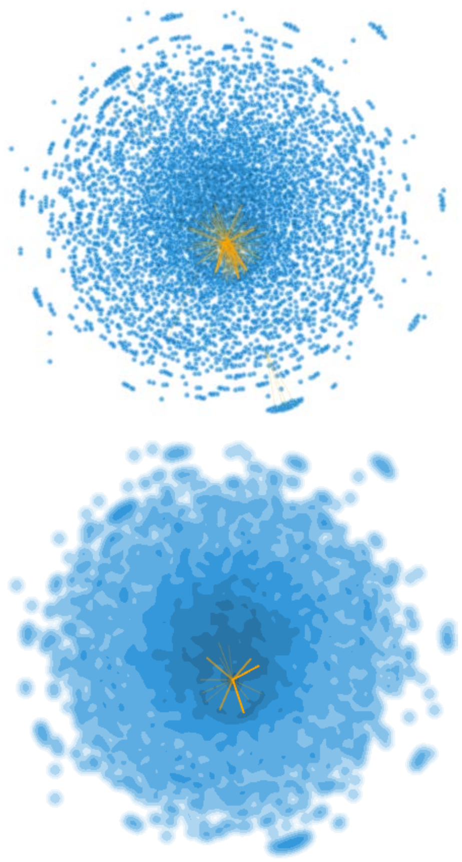

FIGURE 4. **The same network layout with two different vertex aggregation levels. The network has about 25 000 elements, and the vertex aggregation level has been doubled in the right figure.**

associated with the edge segment (whose color is orange). In terms of implementation, for each vertex v_i we identify its cluster by applying the following process. Initially, let p_i be the pixel representing the position of v_i, and consider the 8-neighborhood of p_i. Let q_i be the pixel of this 8-neighborhood with highest density field value. If the value of p_i is larger than q_i, then p_i will be the center of the cluster and the process halts. Otherwise, we set $p_i = q_i$ and we repeat the process.

C. INTERACTION AND USER INTERFACE

The system provides a user interface with general-purpose interaction features.[5] Once the input graph is loaded, the produced visualization is scaled so to fit entirely within a canvas of fixed size. The user interface makes it possible to obtain coarser or finer vertex and edge aggregations, by suitably modifying the aggregation parameters of the density field and of the hill climbing algorithm. Fig. 4 shows two visualizations obtained by modifying the vertex aggregation level.

A classic zoom and pan feature permits to move the drawing (panning) with respect to the canvas, or to scale it

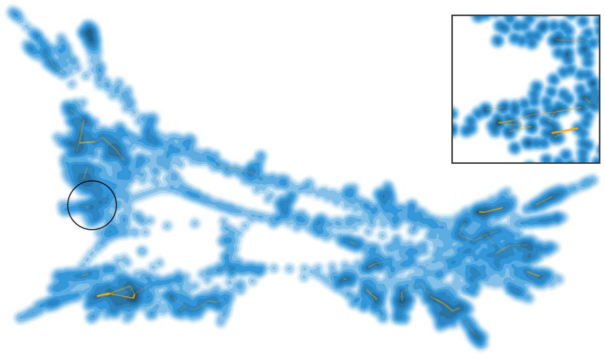

FIGURE 5. **Zooming in a specified portion of the drawing.**

up and down (zooming). As a consequence of a zoom or pan operation, the sets of vertices and edges in the canvas change and both the density field and the edge aggregation are recomputed. In other words, the rendering step is repeated on a portion of the whole layout. When zooming-in, the density field becomes more fine grained and fewer edges are aggregated, while when zooming-out, the density field becomes less detailed and more edges are bundled together. An undesired effect of a quick zoom-in or zoom-out operation is a sudden change in the visualization, which is caused by the sharp (dis)aggregation of vertices and edges. To cope with this issue, an important feature of our interface, is that the vertex and edge aggregation levels are dynamically tuned so to make the whole exploration process more stable. We remark that this feature is not present in [13].

A second feature allows the user to select a smaller portion of the drawing, which is rendered inside a dedicated view with a zoom level chosen by the user and independent of the zoom level of the whole drawing; see Fig. 5 for an illustration.

Finally, BrowVis makes it possible to display a certain percentage of the node labels with higher degree; this percentage is automatically set by the system based on the current zoom level and on the current level of vertex aggregation. In particular, to limit the overall visual complexity, one can choose to visualize the labels in a neighborhood of a desired point. Refer to Section V for illustrations and examples of this feature.

IV. EXPERIMENTAL ANALYSIS

In this section we describe the experimental analysis that we executed in order to assess the performance of BrowVis.

A. GRAPH BENCHMARK

We used three different benchmarks of graphs, already exploited in similar experiments (see, e.g., [11]).

- `Real`. It consists of 12 real networks, with up to 1.5 million edges, taken from the Sparse Matrix Collection of the University of Florida,[6] the Stanford Large Networks Dataset Collection,[7] and the Network Data

[5]A demo version with a preloaded network (the one used in Section V) is available at the following URL: http://mozart.diei.unipg.it/montecchiani/browvis/

[6]http://www.cise.ufl.edu/research/sparse/matrices/

[7]http://snap.stanford.edu/data/index.html

Repository[8] [54]. Details about name, type, and structure of these graphs are reported in Table 1. The whole algorithmic pipeline (layout and rendering) is applied on these graphs after the removal of isolated vertices, self-loops, and parallel edges.

- `Synth-Rand`. It contains 18 synthetic random graphs generated with the Erdõs-Rényi model [55]. These graphs are divided into six groups of three graphs each, with size (number of edges) $m \in \{10^4, 5 \cdot 10^4, 10^5, 10^6, 1.5 \cdot 10^6, 2 \cdot 10^6\}$ and density (number of edges divided by number of vertices) in the range [2, 3].
- `Synth-SF`. It contains 18 synthetic scale-free graphs generated with the Barabasi-Albert model [56]. Again, these graphs are divided into six groups of three graphs each, with size (number of edges[9]) $m \in \{10^4, 5 \cdot 10^4, 10^5, 10^6, 1.5 \cdot 10^6, 2 \cdot 10^6\}$ and density in the range [2, 3].

B. EXPERIMENTAL SETTING

We executed the experiments on a MacBook Pro (Mid 2015) laptop equipped with an i7-4870HQ CPU, 16 GB of RAM, and running macOS Big Sur as operating system. Also, for the experiments we used the 96.0.4664.45 version of the 64-bit Google Chrome browser. For each computation, we measured the running time and the memory footprint. For each graph, we repeated the computation three times.

C. RESULTS

Table 2 shows the recorded running time and memory footprint for each of the three benchmarks. The running time is split between the two main steps, layout and rendering. The values are averaged over the three executions. For `Synth-Rand` and `Synth-SF`, the graphs are further grouped based on the number of edges. The standard deviation is also reported.

- `Real`. One can immediately observe that the layout step is about one order of magnitude slower than the rendering step. The relatively small standard deviation values assess a good stability of the algorithms. The smallest network ($\approx 15 \cdot 10^3$ elements) took about 2.5 seconds to be visualized, while the largest one ($\approx 2.5 \cdot 10^6$ elements) took about 13 minutes. Notably, the rendering step took less than 1 second for all graphs with up to about $2 \cdot 10^5$ elements, and about 14 seconds for the largest instance. Concerning the primary memory required by the computations, it ranges from 17 MB for the smallest graph to about 2.6 GB for the largest instance.
- `Synth-Rand`. Again the layout is significantly slower than the rendering. The standard deviation is large, due to the fact that graphs in the same group can have (slightly) different sizes. However, the more uniform structure of the graphs yields faster running times. The smallest instances ($\approx 14 \cdot 10^3$ elements) took about 2.2 seconds to be visualized, those with $\approx 2 \cdot 10^6$ elements took about 8 minutes, while the largest ones ($\approx 2.8 \cdot 10^6$ elements) took about 11 minutes.
- `Synth-SF`. Also for this benchmark, laying out the graph is significantly slower than rendering the layout. The standard deviation is larger than in the previous cases, because graphs in the same group can have (slightly) different sizes and, in addition, their structure is less uniform with this model. The smallest instances ($\approx 14 \cdot 10^3$ elements) took about 2.5 seconds to be visualized, those with $\approx 2.5 \cdot 10^6$ elements took about 8 minutes, while the largest ones ($\approx 2.8 \cdot 10^6$ elements) took about 9 minutes. Overall, the running time and the memory footprint are slightly lower than those obtained for the `Synth-Rand` graphs. The reason may lie in the fact that scale-free graphs often exhibit a dense core, which results in better aggregations (especially in the rendering step), which in turn yield faster computation times and more compact data structures.

[8] http://www.networkrepository.com/

[9] For each sample m, the actual number of edges of a graph in this sample is approximately m, as the generator does not allow us fixing the number of edges exactly.

D. DISCUSSION

Our experiments show that BrowVis is able to visualize graphs with several thousand edges in seconds, while it can scale up to graphs with millions of edges in minutes. We remark that, once the initial visualization has been computed, any further interaction only requires to (partially) repeat the rendering step, which never took more than 15 seconds in our experiments, and it actually took less than 0.5 seconds for all instances with less than 10^5 edges. In terms of scalability, it shall be noticed that, since the WebAssembly code runs in a sandbox, the communication with JavaScript is obtained via shared memory locations references with 32-bit pointers, which limits the maximum amount of primary memory that can be used to 4GB. Hence, scaling to much larger graphs would require to overcome this memory limit, e.g., by filtering the graph in a preprocessing routine [57].

As already discussed, the experiments conducted in [18] report neither the running time nor the memory footprint of the algorithms. In particular, it is not clear what it is the initial time to wait until a first stable visualization is produced. Yet, the experiments in [18] suggest that BrowVis outperforms the considered technologies, namely Stardust.js and D3.js can render up to a total of 10^5 elements (both vertices and edges), while NetV.js can render up to a total of 10^6 elements showing at least 1 frame per second. In Section V, we will also report about the observed framerate during an interaction session with the computed visualizations. In addition, we remark that BrowVis produces an interactive abstraction of the input graph, which allows for a details-on-demand exploration that avoids clutter and over-plotting.

V. CASE STUDY

In this section, we describe a case study that shows how BrowVis can be effectively exploited to extract useful

TABLE 1. Details for the `Real` benchmark. $|V|$ and $|E|$ are the number of vertices and edges of the instances.

NAME	$\|V\|$	$\|E\| \downarrow$	DESCRIPTION
add32	4 960	9 462	circuit simulation problem
ca-GrQc	5 242	14 496	collaboration network
poli_large	15 575	17 427	chemical process simulation
p2p-Gnutella04	10 876	39 994	P2P network
pGp-giantcomp	10 680	48 632	Pretty-Good-Privacy network
ca-CondMat	23 133	93 497	collaboration network
p2p-Gnutella31	62 586	147 892	P2P network
ASIC_320ks	321 523	515 300	circuit simulation problem
amazon0302	262 111	899 792	co-purchasing network
com-amazon	334 863	925 872	co-purchasing network
com-DBLP	317 080	1 049 866	collaboration network
roadNet-PA	1 087 562	1 541 514	road network

information from a large real-world graph. The input is a scientific collaboration network representing co-authorships in the area of visualization and computer graphics. Specifically, from the popular DBLP repository [58], we collected meta-information about the articles published until 2020 on three journals that are highly representative of the aforementioned research area, namely IEEE Transactions on Visualization and Computer Graphics, IEEE Computer Graphics and Applications, and Computer Graphics Forum. Additionally, we considered all articles appeared until 2020 in the proceedings of the annual Symposium on Graph Drawing and Network Visualization (GD), which focuses on theoretical and application aspects of graph visualization.

From the collected set of articles, we constructed a network such that each node represents an author and two nodes are connected by an edge if they are co-authors of at least one article. The network consists of $n = 16\,435$ nodes and $m = 52\,778$ edges, hence it has density $\frac{m}{n} = 3.2$. In terms of connectivity, the network is composed of 1 293 connected components, 436 of them being isolated nodes. The largest connected component, denoted as C, contains $n_C = 13\,008$ nodes (about 79.1% of the total node set) and $m_C = 48\,464$ edges (about 91.8% of the total edge set); its density is therefore $\frac{m_C}{n_C} = 3.7$.

We visually inspected C with BrowVis, on the same machine used for running the experiments of Section IV: The layout time was of 6.9 seconds and the rendering time was of 0.75 seconds. Without interaction the visualization consists of 60 FPS (frame per second). With the default level of vertex and edge aggregation, the framerate during interaction (zoom or panning) was about 35 FPS. The framerate dropped below 10 FPS with the maximum level of vertex and edge aggregation, which requires the highest computational effort.

Fig. 6 depicts a high-level visualization of the network. The vertex and edge aggregations at this level clearly reveal the presence of three denser areas in the network, loosely connected to each other: Two bigger areas at the core of the layout and a smaller area in the top-left part of the layout. Each area appears as a darker blue zone, which reflects the aggregation of many vertices, containing some thick yellow segments, representing the aggregation of several edges. We inspected the three areas separately to get insights about what they represent. To this aim, we displayed some node labels inside them. Recall that BrowVis allows us to display a certain percentage of the labels of the nodes with higher degree, and that this percentage is automatically decided based on the current zoom level and on the current level of vertex aggregation. At the same time, the user can choose to visualize the labels in a neighborhood of a desired point.

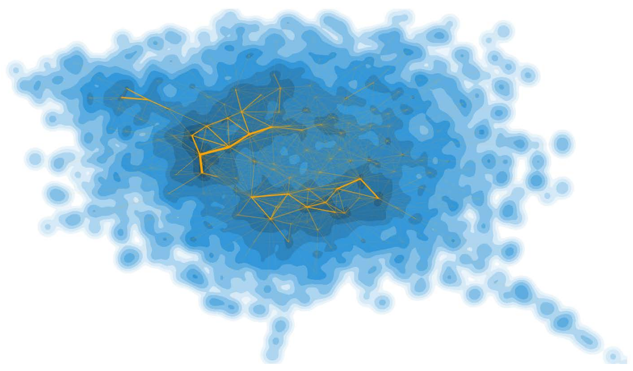
FIGURE 6. High-level view of the scientific collaboration network analyzed in the case study. The visualization reveals three main denser areas, two in the central part of the layout and one smaller in the top-left part.

Using these functionalities, we could observe that each of the three areas refers to a specific type of community. Namely, one of the two big central areas (the top one) consists of authors that mainly work in information visualization and visual analytics; as shown in Fig. 7, some representative authors in this area include K. Ma, H. Qu, H. Pfister, B. Preim, D. A. Keim, and D. S. Ebert. The lower big central area is more focused on computer graphics, with representative authors including H. Seidel and M.H. Gross. Also, some

TABLE 2. Running time and memory footprint. The symbol σ denotes the standard deviation.

		LAYOUT		RENDERING		TOTAL			
		TIME [MS]		TIME [MS]		TIME [MS]		MEMORY [MB]	
	GRAPH NAME	MEAN	σ	MEAN	σ	MEAN	σ	MEAN	σ
Real	add32	2 406	45	185	2	2 591	47	17	0.1
	ca-GrQc	2 440	41	233	4	2 673	44	17	0.2
	poli_large	6 942	69	326	15	7 268	78	41	0.3
	p2p-Gnutella04	6 473	26	312	6	6 785	23	30	0.1
	pGp-giantcomp	6 788	79	303	15	7 091	93	30	0
	ca-CondMat	12 798	38	435	29	13 233	66	60	0.3
	p2p-Gnutella31	40 750	397	956	7	41 707	397	150	0
	ASIC_320ks	210 625	518	3 747	59	214 373	467	747	0.5
	amazon0302	160 928	849	3 620	33	164 548	860	620	0.5
	com-Amazon	220 228	298	4 505	49	224 793	285	788	0.5
	com-DBLP	214 869	1 146	4 479	78	219 349	1 219	753	0.5
	roadNet-PA	778 083	18 037	14 110	123	792 193	17 957	2 631	79
	# EDGES								
Synth-Rand	10 000	2 163	137	219	9	2 383	145	14	1
	50 000	14 229	2 222	479	25	14 708	2 243	57	4
	100 000	26 735	7 686	720	82	27 455	7 768	99	14
	1 000 000	371 526	50 525	6 469	129	377 996	50 646	1 116	12
	1 500 000	501 533	66 731	8 850	826	510 383	67 556	1 507	153
	2 000 000	644 480	40 911	11 653	464	656 133	41 358	2 041	192
Synth-SF	10 000	2 305	316	192	10	2 497	321	15	1
	50 000	10 992	1 074	339	24	11 391	1 098	52	5
	100 000	26 925	5 466	684	55	27 609	5 515	101	9
	1 000 000	326 749	99 959	6 329	1 573	333 078	101 529	1 144	297
	1 500 000	515 796	98 497	9 336	1 473	525 132	99 967	1 677	300
	2 000 000	541 610	31 099	10 055	387	551 665	31 486	1 708	59

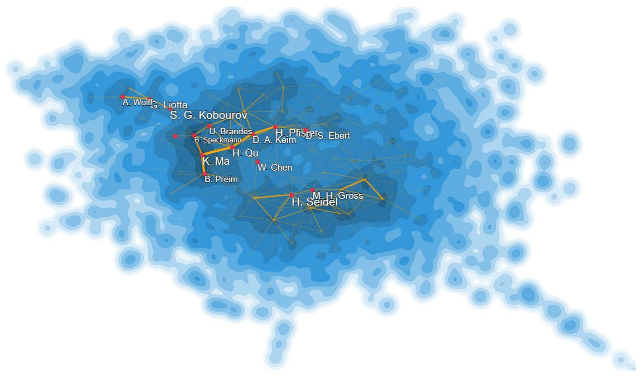

FIGURE 7. Visualization of few representative labels in the different areas. Some authors look as bridges between different communities.

authors (e.g., W. Chen), fall in the middle of the two areas. The smaller area in the top-left part of the network consists of authors in the graph drawing community. It is worth observing that some of them, such as S. G. Kobourov and U. Brandes, are well recognized both in the graph drawing and in the information visualization fields, thus representing a bridge between the two communities. In the layout, these authors are correctly located in the middle of the two corresponding areas.

Fig. 8 is a zoom-in of a portion of the graph drawing community, where more elements are visible. Although the edge and vertex aggregation mechanisms make it often difficult to visually establish the existence of connections between specific pairs of authors, the overall picture provides useful information. For example, it is worth observing that there is a high concentration of edges emanating from some clusters, in particular the one represented by M. Kaufmann, who is in fact recognized as one of the most collaborative authors of this community. The node labels shown in the figure are the authors that share some articles with M. Kaufmann. Also,

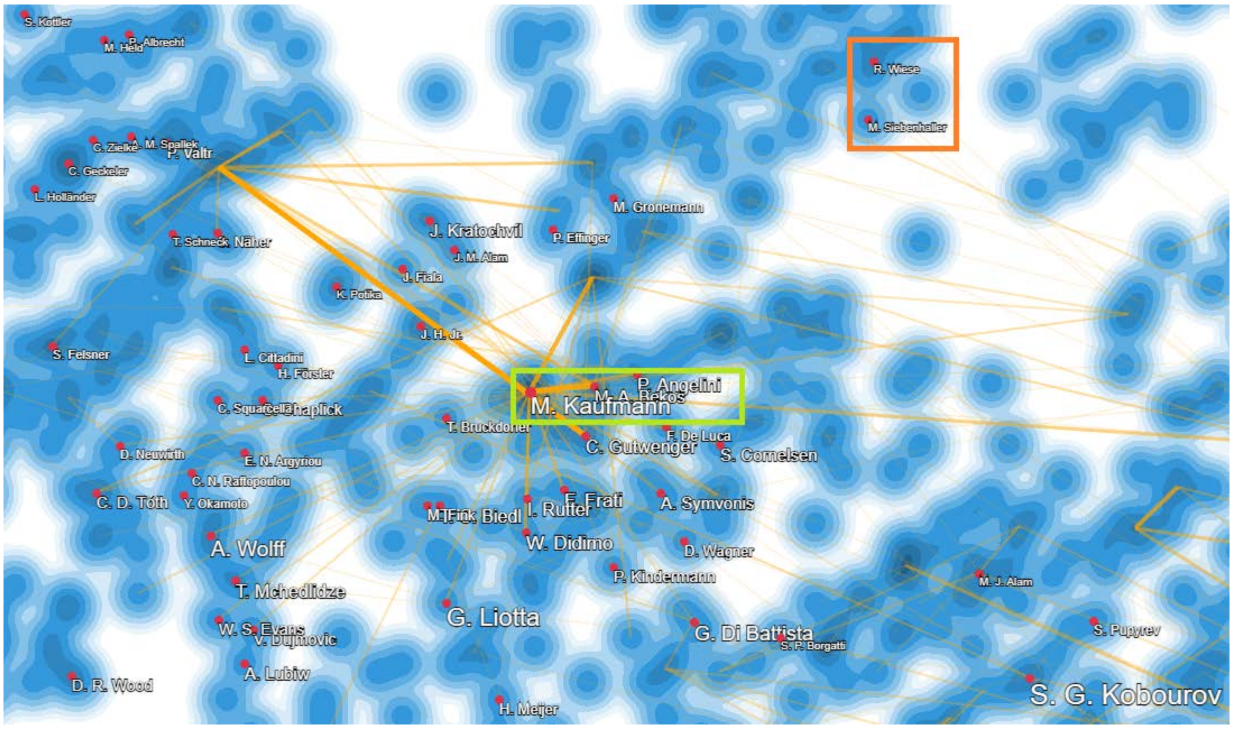

FIGURE 8. Zoom-in of a portion of the graph drawing community. The green highlighted rectangle indicates an intensive and continuous collaboration of Kaufmann with other two authors; the orange rectangle indicates authors that stopped their collaboration with Kaufmann several years ago.

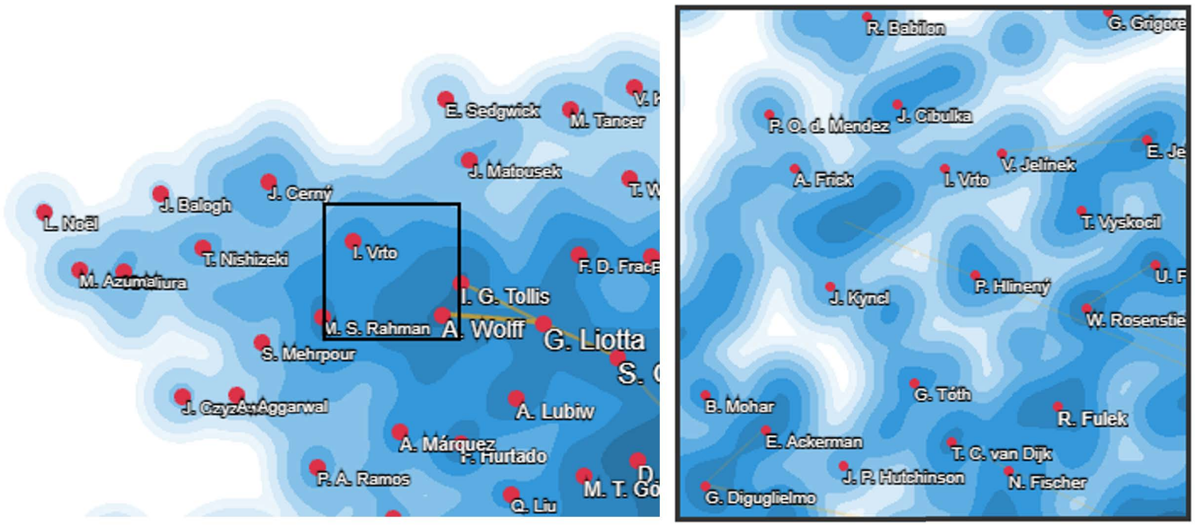

FIGURE 9. Magnification of an upper left portion of the graph drawing community, which highlights scientists working on theoretical aspects of graph drawing.

node proximity in the layout provides additional insights. For instance, the two authors (M. Bekos and P. Angelini) that are very close to M. Kaufmann (see the highlighted green rectangle in Fig. 8) have been collaborating intensively with him for many years; conversely, the farther two authors (R. Wiese and M. Siebenhaller) in the highlighted orange rectangle stopped their collaboration with Kaufmann back in time.

We finally report in Fig. 9 a detail of an upper left portion of the graph drawing community, obtained through the magnifying glass offered by the system. The authors appearing in this area are Eastern European scientists working on theoretical aspects of graph drawing.

VI. CONCLUSION AND FUTURE WORK

Motivated by the goal of exploring the scalability limits of a system that executes the entire graph visualization pipeline in the browser, we developed BrowVis, a self-contained Javascript-based system to interactively visualize large graphs. BrowVis adopts a multi-level force-directed algorithm to layout the input graph, and it allows an interactive exploration of the produced visualization, with adjustable level of abstraction. A publicly available proof-of-concept implementation of BrowVis has been used to run an extensive experimental analysis. On a common laptop, BrowVis can visualize graphs with several thousand elements in seconds, as well as graphs with millions of elements in minutes. In addition, the usefulness of BrowVis is demonstrated in a visual exploration session of a large real-world scientific collaboration network.

Our work demonstrates that visualization pipelines of considerably large graphs can be entirely integrated in client-side systems. Still, there are several research directions that are worth pursuing. Among them:

- The most natural research direction is to further speed-up our techniques. We believe that pursuing such a direction, especially for the rendering step, requires the design of more sophisticated data structures to update the density field and the edge bundles dynamically. In addition, alternative exploration paradigms or visual abstraction methods may contribute to this research direction.
- We focused on static graphs whose structure does not change over time. Dealing with dynamic graphs, such as those originated by streaming data sources, would open the way to new interesting applications (see, e.g., [22]). This would require to re-think both the layout step, which should produce layouts that remain stable over time, as well as the rendering step, which should highlight the changes that occur over time.
- Integrating intelligent agents that employ task offloading strategies [59] is also of great interest, in order to exploit a broader and dynamic range of available computing resources, spanning from local hardware to the cloud through edge devices.
- Our user interface is designed for laptop and desktop screens. Optimizing it for smartphones would be useful. In this direction, the study of intelligent interfaces that dynamically optimize and adjust their menus represents an active line of research [60].

ACKNOWLEDGMENT

This is an expanded paper from the First Italian Conference on Big Data and Data Science (ITADATA 2022).

REFERENCES

[1] X. Jia, T. Wen, W. Ding, H. Li, and W. Li, "Semi-supervised label distribution learning via projection graph embedding," *Inf. Sci.*, vol. 581, pp. 840–855, Dec. 2021.

[2] W. Didimo, L. Grilli, G. Liotta, F. Montecchiani, and D. Pagliuca, "Visual querying and analysis of temporal fiscal networks," *Inf. Sci.*, vol. 505, pp. 406–421, Dec. 2019.

[3] Z. Lin, L. Feng, R. Yin, C. Xu, and C. K. Kwoh, "GLIMG: Global and local item graphs for top-N recommender systems," *Inf. Sci.*, vol. 580, pp. 1–14, Nov. 2021.

[4] N. Binesh and M. Ghatee, "Distance-aware optimization model for influential nodes identification in social networks with independent cascade diffusion," *Inf. Sci.*, vol. 581, pp. 88–105, Dec. 2021.

[5] R. Baggio and M. Fuchs, "Network science and E-tourism," *Inf. Technol. Tourism*, vol. 20, nos. 1–4, pp. 97–102, Dec. 2018.

[6] L. T. Mohammed, A. A. AlHabshy, and K. A. ElDahshan, "Big data visualization: A survey," in *Proc. HORA*, 2022, pp. 1–12.

[7] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu, "The ubiquity of large graphs and surprising challenges of graph processing: Extended survey," *VLDB J.*, vol. 29, nos. 2–3, pp. 595–618, May 2020.

[8] S.-H. Cheong and Y.-W. Si, "Force-directed algorithms for schematic drawings and placement: A survey," *Inf. Visualizat.*, vol. 19, no. 1, pp. 65–91, Jan. 2020.

[9] Y. Hu and L. Shi, "Visualizing large graphs," *Wiley Interdiscipl. Rev., Comput. Statist.*, vol. 7, no. 2, pp. 115–136, Mar. 2015.

[10] S. G. Kobourov, "Force-directed drawing algorithms," in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed. Boca Raton, FL, USA: CRC Press, 2013.

[11] A. Arleo, W. Didimo, G. Liotta, and F. Montecchiani, "Large graph visualizations using a distributed computing platform," *Inf. Sci.*, vol. 381, pp. 124–141, Mar. 2017.

[12] A. Arleo, W. Didimo, G. Liotta, and F. Montecchiani, "A distributed multilevel force-directed algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 754–765, Apr. 2019.

[13] M. Zinsmaier, U. Brandes, O. Deussen, and H. Strobelt, "Interactive level-of-detail rendering of large graphs," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 12, pp. 2486–2495, Dec. 2012.

[14] K. El Gebaly and J. Lin, "In-browser interactive SQL analytics with afterburner," in *Proc. ACM Int. Conf. Manag. Data*, May 2017, pp. 1623–1626.

[15] J. Lin, "Building a self-contained search engine in the browser," in *Proc. Int. Conf. Theory Inf. Retr.*, Sep. 2015, pp. 309–312.

[16] J. Lee, R. Tang, and J. Lin, "Honkling: In-browser personalization for ubiquitous keyword spotting," in *Proc. Conf. Empirical Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process. (EMNLP-IJCNLP)*, 2019, pp. 91–96.

[17] S. Dutta and S. Roy, "Complex network visualisation using JavaScript: A review," in *Intelligent Systems*, S. K. Udgata, S. Sethi, and X.-Z. Gao, Eds. Singapore: Springer, 2022, pp. 45–53.

[18] D. Han, J. Pan, X. Zhao, and W. Chen, "NetV.Js: A web-based library for high-efficiency visualization of large-scale graphs and networks," *Vis. Informat.*, vol. 5, no. 1, pp. 61–66, Mar. 2021.

[19] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel, "The open graph drawing framework (OGDF)," in *Handbook of Graph Drawing and Visualization*. Boca Raton, FL, USA: CRC Press, 2013, pp. 543–569.

[20] S. Hachul and M. Junger, "Large-graph layout algorithms at work: An experimental study," *J. Graph Algorithms Appl.*, vol. 11, no. 2, pp. 345–369, 2007.

[21] T. Von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner, "Visual analysis of large graphs: State-of-the-art and future research challenges," *Comput. Graph. Forum*, vol. 30, no. 6, pp. 1719–1749, Sep. 2011.

[22] S.-H. Cheong, Y.-W. Si, and R. K. Wong, "Online force-directed algorithms for visualization of dynamic graphs," *Inf. Sci.*, vol. 556, pp. 223–255, May 2021.

[23] W. Didimo and F. Montecchiani, "Fast layout computation of clustered networks: Algorithmic advances and experimental analysis," *Inf. Sci.*, vol. 260, pp. 185–199, Mar. 2014.

[24] U. Dogrusoz, E. Giral, A. Cetintas, A. Civril, and E. Demir, "A layout algorithm for undirected compound graphs," *Inf. Sci.*, vol. 179, no. 7, pp. 980–994, Mar. 2009.

[25] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Softw., Pract. Exper.*, vol. 21, no. 11, pp. 1129–1164, Nov. 1991.

[26] A. J. Quigley and P. Eades, "FADE: Graph drawing, clustering, and visual abstraction," in *Graph Drawing* (Lecture Notes in Computer Science), vol. 1984. Berlin, Germany: Springer, 2000, pp. 197–210.

[27] R. Hadany and D. Harel, "A multi-scale algorithm for drawing graphs nicely," *Discrete Appl. Math.*, vol. 113, no. 1, pp. 3–21, Sep. 2001.

[28] C. Walshaw, "A multilevel algorithm for force-directed graph-drawing," *J. Graph Algorithms Appl.*, vol. 7, no. 3, pp. 253–285, 2003.

[29] S. Hachul, "A potential field based multilevel algorithm for drawing large graphs," Ph.D. dissertation, Mathematisch-Naturwissenschaftlichen Fakultaet, Univ. Cologne, Cologne, Germany, 2005. [Online]. Available: http://kups.ub.uni-koeln.de/volltexte/2005/1409/index.html

[30] S. Hachul and M. Jünger, "Drawing large graphs with a potential-field-based multilevel algorithm," in *Graph Drawing* (Lecture Notes in Computer Science), vol. 3383. Berlin, Germany: Springer, 2004, pp. 285–295.

[31] G. Bartel, C. Gutwenger, K. Klein, and P. Mutzel, "An experimental evaluation of multilevel layout methods," in *Graph Drawing* (Lecture Notes in Computer Science), vol. 6502. Berlin, Germany: Springer, 2010, pp. 80–91.

[32] D. Auber and Y. Chiricota, "Improved efficiency of spring embedders: Taking advantage of GPU programming," in *VIIP*. Palma de Mallorca, Spain: ACTA Press, 2007, pp. 169–175.
[33] A. Godiyal, J. Hoberock, M. Garland, and J. C. Hart, "Rapid multipole graph drawing on the GPU," in *Graph Drawing* (Lecture Notes in Computer Science), vol. 5417. Berlin, Germany: Springer, 2009, pp. 90–101.
[34] S. Ingram, T. Munzner, and M. Olano, "Glimmer: Multilevel MDS on the GPU," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 2, pp. 249–261, Mar./Apr. 2009.
[35] E. Yunis, R. Yokota, and A. Ahmadia, "Scalable force directed graph layout algorithms using fast multipole methods," in *Proc. 11th Int. Symp. Parallel Distrib. Comput.*, Jun. 2012, pp. 180–187.
[36] H. Meyerhenke, M. Nollenburg, and C. Schulz, "Drawing large graphs by multilevel maxent-stress optimization," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 5, pp. 1814–1827, May 2018.
[37] C. Mueller, D. Gregor, and A. Lumsdaine, "Distributed force-directed graph layout and visualization," in *Proc. EGPGV*, Eurographics, 2006, pp. 83–90.
[38] S. Chae, A. Majumder, and M. Gopi, "HD-GraphViz: Highly distributed graph visualization on tiled displays," in *Proc. 8th Indian Conf. Comput. Vis., Graph. Image Process. (ICVGIP)*, 2012, pp. 43.1–43.8.
[39] A. Tikhonova and K. Ma, "A scalable parallel force-directed graph layout algorithm," in *Proc. EGPGV*, Eurographics, 2008, pp. 25–32.
[40] A. Hinge and D. Auber, "Distributed graph layout with spark," in *Proc. 19th Int. Conf. Inf. Visualisation*, 2015, pp. 271–276.
[41] A. Hinge, G. Richer, and D. Auber, "MuGDAD: Multilevel graph drawing algorithm in a distributed architecture," in *Proc. Conf. Comput. Graph., Vis. Comput. Vis.*, 2017, p. 189.
[42] A. Meidiana, S.-H. Hong, S. Cai, M. Torkel, and P. Eades, "Sublinear-time attraction force computation for large complex graph drawing," in *Proc. IEEE 14th Pacific Visualizat. Symp. (PacificVis)*, Apr. 2021, pp. 146–150.
[43] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proc. IEEE Symp. Vis. Lang.*, Apr. 1996, pp. 336–343.
[44] J. Abello, F. van Ham, and N. Krishnan, "ASK-GraphView: A large scale graph visualization system," *IEEE Trans. Vis. Comput. Graphics*, vol. 12, no. 5, pp. 669–676, Sep. 2006.
[45] D. Auber, Y. Chiricota, F. Jourdan, and G. Melancon, "Multiscale visualization of small world networks," in *Proc. IEEE Symp. Inf. Visualizat.*, Oct. 2003, pp. 75–81.
[46] F. van Ham and J. J. van Wijk, "Interactive visualization of small world graphs," in *Proc. IEEE Symp. Inf. Visualizat.*, Oct. 2004, pp. 199–206.
[47] A. Perrot and D. Auber, "Cornac: Tackling huge graph visualization with big data infrastructure," *IEEE Trans. Big Data*, vol. 6, no. 1, pp. 80–92, Mar. 2020.
[48] M. Franz, C. T. Lopes, G. Huck, Y. Dong, S. O. Sümer, and G. D. Bader, "Cytoscape.Js: A graph theory library for visualisation and analysis," *Bioinformatics*, vol. 32, no. 2, pp. 309–311, 2016.
[49] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.
[50] J.-P. Coene, "Sigmajs: An R htmlwidget interface to the sigma. Js visualization library," *J. Open Source Softw.*, vol. 3, no. 28, p. 814, Aug. 2018.
[51] D. Ren, B. Lee, and T. Höllerer, "Stardust: Accessible and transparent GPU support for information visualization rendering," *Comput. Graph. Forum*, vol. 36, no. 3, pp. 179–188, Jun. 2017.
[52] C. Rodriguez, P. Toharia, L. Pastor, and S. Mata, "Carbonic: A framework for creating and visualizing complex compound graphs," *Appl. Sci.*, vol. 12, no. 15, p. 7541, Jul. 2022.
[53] F. Auer, S. Mayer, and F. Kramer, "Data-dependent visualization of biological networks in the web-browser with ndexedit," *PLOS Comput. Biol.*, vol. 18, no. 6, pp. 1–13, Jun. 2022.
[54] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proc. AAAI*, 2015, pp. 4292–4293. [Online]. Available: http://networkrepository.com
[55] P. Erdos and A. Rényi, "On random graphs I," *Pub. Math. (Debrecen)*, vol. 6, pp. 290–297, Dec. 1959.
[56] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
[57] X. Huang and C. Huang, "NGD: Filtering graphs for visual analysis," *IEEE Trans. Big Data*, vol. 4, no. 3, pp. 381–395, Sep. 2018.
[58] M. Ley, "The DBLP computer science bibliography: Evolution, research issues, perspectives," in *String Processing and Information Retrieval* (Lecture Notes in Computer Science), vol. 2476. Berlin, Germany: Springer, 2002, pp. 1–10.
[59] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, A. Leivadeas, N. Athanasopoulos, N. Mitton, and S. Papavassiliou, "Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions," *Comput. Netw.*, vol. 195, Aug. 2021, Art. no. 108177.
[60] G. Pau, F. Arena, M. Collotta, and X. Kong, "A practical approach based on Bluetooth low energy and neural networks for indoor localization and targeted devices identification by smartphones," *Entertainment Comput.*, vol. 43, Aug. 2022, Art. no. 100512.

LUCA CONSALVI received the master's degree in computer and robotics engineering from the University of Perugia, in 2021. While developing and writing his thesis, he studied web technologies and information visualization techniques for network visualization and network analysis.

WALTER DIDIMO received the Ph.D. degree in computer engineering from the University of Rome "La Sapienza," in 2000. He is currently an Associate Professor at the Department of Engineering, University of Perugia. His research interests include graph drawing, information visualization, algorithm engineering, and computational geometry. He collected more than 150 international publications in the above areas and chaired the program committee of the International Symposium on Graph Drawing. He currently serves as an Associate Editor for the IEEE Access journal.

GIUSEPPE LIOTTA (Senior Member, IEEE) is currently a Professor at the Department of Engineering, University of Perugia, Italy. His research interests include information visualization, graph drawing, and computational geometry. On these topics, he published more than 250 research papers. He chaired the Steering Committee of the International Symposium of Graph Drawing and Network Visualization and he currently serves as the Editor-in-Chief of *Computer Science Review* and the *Journal of Graph Algorithms and Applications*.

FABRIZIO MONTECCHIANI received the Ph.D. degree in information engineering from the University of Perugia, in 2014. He currently works as an Associate Professor with the University of Perugia. His research interests include graph drawing, computational geometry, visual analytics, and big data algorithms. He collected more than 100 scientific publications in the above areas. In 2021, he has been awarded by the IC-EATCS as the Best Young Italian Researcher in Theoretical Computer Science. He has been the Guest Editor of the *Journal of Graph Algorithms & Applications* and *Future Internet*, and he has been a PC Member of international conferences, such as EuroCG, GD, MFCS, and WG.

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