A Distributed Multilevel Force-Directed Algorithm

Alessio Arleo(✉), Walter Didimo(✉), Giuseppe Liotta, and Fabrizio Montecchiani

Università Degli Studi di Perugia, Perugia, Italy
alessio.arleo@studenti.unipg.it,
{walter.didimo,giuseppe.liotta,fabrizio.montecchiani}@unipg.it

Abstract. The wide availability of powerful and inexpensive cloud computing services naturally motivates the study of distributed graph layout algorithms, able to scale to very large graphs. Nowadays, to process Big Data, companies are increasingly relying on PaaS infrastructures rather than buying and maintaining complex and expensive hardware. So far, only a few examples of *basic* force-directed algorithms that work in a distributed environment have been described. Instead, the design of a distributed *multilevel* force-directed algorithm is a much more challenging task, not yet addressed. We present the first multilevel force-directed algorithm based on a distributed vertex-centric paradigm, and its implementation on Giraph, a popular platform for distributed graph algorithms. Experiments show the effectiveness and the scalability of the approach. Using an inexpensive cloud computing service of Amazon, we draw graphs with ten million edges in about 60 min.

1 Introduction

Force-directed algorithms are very popular techniques to automatically compute graph layouts. They model the graph as a physical system, where attractive and repulsive forces act on each vertex. Computing a drawing corresponds to finding an equilibrium state (i.e., a state of minimum energy) of the force system through a simple iterative approach. Different kinds of force and energy models give rise to different graph drawing algorithms. Refer to the work of Kobourov for a survey on the many force-directed algorithms described in the literature [25]. Although basic force-directed algorithms usually compute nice drawings of small or medium graphs, using them to draw large graphs has two main obstacles: (*i*) There could be several local minima in their physical models: if the algorithm falls in one of them, it may produce bad drawings. The probability of this event and its negative effect increase with the size of the graph. (*ii*) Their approach is computationally expensive, thus it gives rise to scalability problems even for graphs with a few thousands of vertices.

Research supported in part by the MIUR project AMANDA "Algorithmics for MAssive and Networked DAta", prot. 2012C4E3KT_001.

Y. Hu and M. Nöllenburg (Eds.): GD 2016, LNCS 9801, pp. 3–17, 2016.
DOI: 10.1007/978-3-319-50106-2_1

To overcome the above obstacles, *multilevel* force-directed algorithms have been conceived. A limited list of works on this subject includes [13,15,18,20,21,23,33] (see [25] for more references). These algorithms generate from the input graph G a series (hierarchy) of progressively simpler structures, called *coarse graphs*, and then incrementally compute a drawing of each of them in reverse order, from the simplest to the most complex (corresponding to G). On common machines, multilevel force-directed algorithms perform quickly on graphs with several thousand vertices and usually produce qualitatively better drawings than basic algorithms [8,19,25]. Implementations based on GPUs have been also experimented [16,24,29,34]. They scale to graphs with a few million edges, but their development requires a low-level implementation and the necessary infrastructure could be expensive in terms of hardware and maintenance.

The wide availability of powerful and inexpensive cloud computing services and the growing interest towards PaaS infrastructures observed in the last few years, naturally motivate the study of distributed graph layout algorithms, able to scale to very large graphs. So far, the design of distributed graph visualization algorithms has been only partially addressed. Mueller *et al.* [27] and Chae *et al.* [9] proposed 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; scalability experiments are limited to graphs with some thousand vertices. Tikhonova and Ma [30] presented a parallel force-directed algorithm that can run on graphs with few hundred thousand edges. It takes about 40 minutes for a graph of 260,385 edges, on 32 processors of the PSC's BigBen Cray XT3 cluster. More recently, the use of emerging frameworks for distributed graph algorithms has been investigated. Hinge and Auber [22] described a distributed basic force-directed algorithm implemented in the Spark framework, using the GraphX library. Their algorithm is mostly based on a MapReduce paradigm and shows margins for improvement: it takes 5 hours on a graph with 8,000 vertices and 35,000 edges, on a cluster of 16 machines, each equipped with 24 cores and 48 GB of RAM. A distributed basic force-directed algorithm running on the Apache Giraph framework has been presented in [7] (see also [5] for an extended version of this work). Giraph is a popular platform for distributed graph algorithms, based on a vertex-centric paradigm, also called the TLAV ("Think Like a Vertex") paradigm [11]. Giraph is used by Facebook to analyze the huge network of its users and their connections [12]. The algorithm in [7] can draw graphs with a million edges in a few minutes, running on an inexpensive cloud computing infrastructure. However, the design of a distributed multilevel force-directed algorithm is a much more challenging task, due to the difficulty of efficiently computing the hierarchy required by a multilevel approach in a distributed manner (see, also [5,22]).

Our Contribution. This paper presents Multi-GiLA (Multilevel Giraph Layout Algorithm), the first distributed multilevel force-directed algorithm based on the TLAV paradigm and running on Giraph. The model for generating the coarse graph hierarchy is inspired by FM3 (Fast Multipole Multilevel Method), one

of the most effective multilevel techniques described in the literature [8,18,19]. The basic force-directed algorithm used by Multi-GiLA to refine the drawing of each coarse graph is the distributed algorithm in [5] (Sect. 3). We show the effectiveness and the efficiency of our approach by means of an extensive experimental analysis: Multi-GiLA can draw graphs with ten million edges in about 60 min (see Sect. 4), using an inexpensive PaaS of Amazon, and exhibits high scalability. To allow replicability of the experiments, our source code and graph benchmarks are made publicly available [1]. It is worth observing that in order to get an overview of the structure of a very large graph and subsequently explore it in more details, one can combine the use of Multi-GiLA with systems like LaGo [35], which provides an interactive level-of-detail rendering, conceived for the exploration of large graphs (see Sect. 4). Section 2 contains the necessary background on multilevel algorithms and on Giraph. Conclusions and future research are in Sect. 5. Additional figures can be found in [6].

2 Background

Multilevel Force-Directed Algorithms. Multilevel force-directed algorithms work in three main phases: *coarsening*, *placement*, and *single-level layout*. Given an input graph G, the coarsening phase computes a sequence of graphs $\{G = G_0, G_1, \ldots, G_k\}$, such that the size of G_{i+1} is smaller than the size of G_i, for $i = 0, \ldots, k-1$. To compute G_{i+1}, subsets of vertices of G_i are merged into single vertices. The criterion for deciding which vertices should be merged is chosen as a trade-off between two conflicting goals. On one hand, the overall graph structure should be preserved throughout the sequence of graphs, as it influences the way the graph is unfolded. On the other hand, both the number of graphs in the sequence and the size of the coarsest graph may have a significant influence on the overall running time of the algorithm. Therefore, it is fundamental to design a coarsening phase that produces a sequence of graphs whose sizes quickly decrease, and, at the same time, whose structures smoothly change. The sequence of graphs produced by the coarsening phase is then traversed from G_k to $G_0 = G$, and a final layout of G is obtained by progressively computing a layout for each graph in the sequence. In the placement phase, the vertices of G_i are placed by exploiting the information of the (already computed) drawing Γ_{i+1} of G_{i+1}. Starting from this initial placement, in the single-level (basic) layout phase, a drawing Γ_i of G_i is computed by applying a single-level force-directed algorithm. Thanks to the good initial placement, such an algorithm will reach an equilibrium after a limited number of iterations. For G_k an initial placement is not possible, thus the layout phase is directly applied starting from a random placement.

Since our distributed multilevel force-directed algorithm is partially based on the FM3 algorithm, we briefly recall how the coarsening and placement phases are implemented by FM3 (see [17,18] for details). Let $G = G_0$ be a connected graph (distinct connected components can be processed independently), the coarsening phase is implemented through the Solar Merger algorithm. The vertices of G are partitioned into vertex-disjoint subgraphs called *solar systems*. The diameter of each solar system is at most four. Within each solar

system S, there is a vertex s classified as a *sun*. Each vertex v of S at distance one (resp., two) from s is classified as a *planet* (resp., a *moon*) of S. There is an *inter-system link* between two solar systems S_1 and S_2, if there is at least an edge of G between a vertex of S_1 and a vertex of S_2. The coarser graph G_1 is obtained by collapsing each solar system into the corresponding sun, and the inter-system links are transformed into edges connecting the corresponding pairs of suns. Also, all vertices of $G = G_0$ are associated with a *mass* equal to one. The mass of a sun is the sum of the masses of all vertices in its solar system. The coarsening procedure halts when a coarse graph has a number of vertices below a predefined threshold. The placement phase of FM3 is called Solar Placer and uses information from the coarsening phase. The vertices of G_{i+1} correspond to the suns of G_i, whose initial position is defined in the drawing Γ_{i+1}. The position of each vertex v in $G_i \setminus G_{i+1}$ is computed by taking into account all inter-system links to which v belongs. The rough idea is to position v in a barycentric position with respect to the positions of all suns connected by an inter-system link that passes through v.

The TLAV Paradigm and the Giraph Framework. The TLAV paradigm requires to implement distributed algorithms from the perspective of a vertex rather than of the whole graph. Each vertex can store a limited amount of data and can exchange messages only with its neighbors. The TLAV framework Giraph [11] is built on the Apache Hadoop infrastructure and originated as the open source counterpart of Google's *Pregel* [26] (based on the *BSP model* [31]). In Giraph, the computation is split into *supersteps* executed iteratively and synchronously. A superstep consists of two phases: (*i*) Each vertex executes a user-defined vertex function based on both local vertex data and on data coming from its adjacent vertices; (*ii*) Each vertex sends the results of its local computation to its neighbors, along its incident edges. The whole computation ends after a fixed number of supersteps or when certain user-defined conditions are met (e.g., no message has been sent or an equilibrium state is reached).

Design Challenges and the GiLA Algorithm. Force-directed algorithms (both single-level and multilevel) are conceived as sequential, shared-memory graph algorithms, and thus are inherently centralized. On the other hand, the following three properties must be guaranteed in the design of a TLAV-based algorithm: `P1.` Each vertex exchange messages only with its neighbors; `P2.` Each vertex locally stores a small amount of data; `P3.` The communication load in each supertsep (number and length of messages sent in the superstep) is small: for example, linear in the number of edges of the graph. Property `P1` corresponds to an architectural constraint of Giraph. Violating `P2` causes out-of-memory errors during the computation of large instances, which translates in the impossibility of storing large routing tables in each vertex to cope with the absence of global information. Violating `P3` quickly leads to inefficient computations, especially on graphs that are locally dense or that have high-degree vertices. Hence, sending heavy messages containing the information related to a large part of the graph is not an option.

In the design of a multilevel force-directed algorithm, the above three constraints P1–P3 do not allow for simple strategies to make a vertex aware of the topology of a large part of the graph, which is required in the coarsening phase. In Sect. 3 we describe a sophisticated distributed protocol used to cope with this issue. For the same reason, a vertex is not aware of the positions of all other vertices in the graph, which is required to compute the repulsive forces acting on the vertex in the single-level layout phase. The algorithm described in [5], called GILA, addresses this last issue by adopting a locality principle, based on the experimental evidence that in a drawing computed by a force-directed algorithm (see, e.g., [25]) the graph theoretic distance between two vertices is a good approximation of their geometric distance, and that the repulsive forces between two vertices u and v tend to be less influential as their geometric distance increases. Following these observations, in the GILA algorithm, the resulting force acting on each vertex v only depends on its *k-neighborhood* $N_v(k)$, i.e., the set of vertices whose graph theoretic distance from v is at most k, for a predefined small constant k. Vertex v acquires the positions of all vertices in $N_v(k)$ by means of a controlled flooding technique. According to an experimental analysis in [5], $k = 3$ is a good trade-off between drawing quality and running time. The attractive and repulsive forces acting on a vertex are defined using Fruchterman-Reingold model [14].

3 The Multi-GILA Algorithm

In this section we describe our multilevel algorithm MULTI-GILA. It is designed having in mind the challenges and constraints discussed in Sect. 2. The key ingredients of MULTI-GILA are a distributed version of both the SOLAR MERGER and of the SOLAR PLACER used by FM3, together with a suitable dynamic tuning of GILA.

3.1 Algorithm Overview

The algorithm is based on the pipeline described below. The pruning, partitioning, and reinsertion phases are the same as for the GILA algorithm, and hence they are only briefly recalled (see [5] for details).

Pruning: In order to lighten the algorithm execution, all vertices of degree one are temporarily removed from the graph; they will be reinserted at the end of the computation by means of an ad-hoc technique.

Partitioning: The vertex set is then partitioned into subsets, each assigned to a computing unit, also called *worker* in Giraph (each computer may have more than one worker). The default partitioning algorithm provided by Giraph may create partitions with a very high number of edges that connect vertices of different partition sets; this would negatively affect the communication load between different computing units. To cope with this problem, we use a partitioning algorithm by Vaquero *et al.* [32], called SPINNER, which creates balanced partition sets by exploiting the graph topology.

Layout: This phase executes the pipeline of the multilevel approach. The coarsening phase (Sect. 3.2) is implemented by means of a distributed protocol, which attempts to behave as the Solar Merger of FM3. The placement (Sect. 3.3) and single-level layout (Sect. 3.4) phases are iterated until a drawing of the graph is computed.

Reinsertion: For each vertex v, its neighbors of degree one (if any) are suitably reinserted in a region close to v, avoiding to introduce additional edge crossings.

This pipeline is applied independently to each connected component of the graph, and the resulting layouts are then arranged in a matrix to avoid overlaps.

3.2 Coarsening Phase: Distributed Solar Merger

Our Distributed Solar Merger algorithm yields results (in terms of number of levels) comparable to those obtained with the Solar Merger of FM3 (see also Sect. 4). The algorithm works into four steps described below; each of them involve several Giraph supersteps. For every iteration i of these four steps, a new coarser graph G_i is generated, until its number of vertices is below a predefined threshold. We use the same terminology as in Sect. 2, and equip each vertex with four properties called *ID*, *level*, *mass*, and *state*. The ID is the unique identifier of the vertex. The level represents the iteration in which the vertex has been generated. That is, a vertex has level i if it belongs to graph G_i. The vertices of the input graph have level zero. The second property represents the mass of the vertex and it is initialized to one plus the number of its previously pruned neighbors of degree one for the vertices of the input graph. The state of a vertex can receive one of the following values: *sun*, *planet*, *moon*, or *unassigned*. We shall call sun, planet, moon, or unassigned, a vertex with the corresponding value for its state. All vertices of the input graph are initially unassigned.

Sun Generation. In the first superstep, each vertex turns its state to sun with probability p, for a predefined value of p. The next three supersteps aim at avoiding pairs of suns with graph theoretic distance less than 3. First, each sun broadcasts a message containing its ID. In the next superstep, if a sun t receives a message from an adjacent sun s, then also s receives a message from t, and the sun between s and t with lower ID changes its state to unassigned. In the same superstep, all vertices (of any state) broadcast to their neighbors only the messages received from those vertices still having state sun. In the third superstep, if a sun t receives a message generated from a sun s (with graph theoretic distance 2 from t), again also t receives a message from s and the sun with lower ID changes its state to unassigned. This procedure ensures that all pairs of suns have graph theoretic distance at least three.

Solar System Generation. In the first superstep, each sun broadcasts an *offer message*. At the next superstep, if an unassigned vertex v receives an offer message m from a sun s, then v turns its state to planet and stores the ID of s in a property called *system-sun*. Also, v sends a *confirmation message* to s. Finally, v forwards the message m to all its neighbors. At the next superstep,

every sun vertex processes the received confirmation messages. If a sun s received a confirmation message, s stores the ID of the sender in a property called *planet-list*. This property is used by each sun to keep track of the planets in its solar system. If a planet v receives an offer message, then such a message comes from the same sun stored in the system-sun property of v, and thus it can be ignored (recall that the theoretic distance between two suns is greater than two). If an unassigned vertex u receives one or more offer messages originated by the same sun s, then u turns its state to moon and stores the ID of s in its system-sun property. Furthermore, u stores the ID of all planets that forwarded the above offer messages in a property called *system-planets*. This property is used by each moon u to keep track of the planets adjacent to u and in the same solar system as u. Finally, u sends a confirmation message to its sun s through a two-hop message (that requires two further supersteps to be delivered), which will be sent to one of the planets stored in the system-planets property. If u receives offer messages from distinct vertices, then the above procedure is applied only for those messages originated by the sun s with greatest ID. For every offer message originated by a sun t with ID lower than the one of s, u informs both s and t of the conflict through ad-hoc two-hop messages. These messages will be used by s and t to maintain a suitable data structure containing the information of each path between s and t. At the end of this phase, all the galaxies of the generated sun vertices have been created and have diameter at most four. Also, some of the inter-system links have already been discovered, and this information will be useful in the following. The two steps described above are repeated until there are no more unassigned vertices. An example is illustrated in Fig. 1.

Inter-system Link Generation. In the first superstep, every planet and every moon broadcasts an *inter-link discovery message* containing the ID stored in the system-sun property of the vertex. In the next superstep, each vertex v processes the received messages. All messages originated by vertices in the same solar system are ignored. Similarly as in the previous step, for each inter-link discovery message originated from a sun t different from the sun s of v, vertex v informs both s and t of the conflict through two-hop messages that will be used

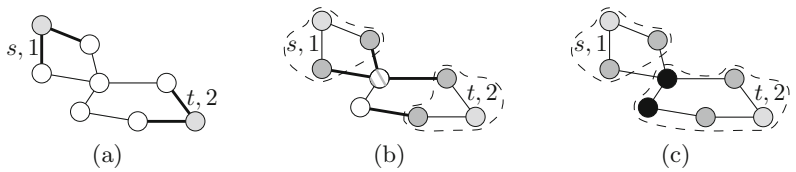


Fig. 1. Illustration for the coarsening phase. (a) Two suns s (ID 1) and t (ID 2) broadcast an offer message. (b) The dark gray vertices receive the offer messages, become planets, and forward the received offer messages. The striped vertex will then receive offer messages from both s and t, and (c) will accept the offer message of t due to the greatest ID of t. In (c) the final galaxies are enclosed by dashed curves, suns (planets, moons) are light gray (dark gray, black).

by s and t to maintain a suitable data structure containing the information of each path between s and t. Once all messages have been delivered, each sun s is aware of all links between its solar system and other systems. Also, for each link, s knows what planet and moon (if any) are involved.

Next Level Generation. In the first superstep, every sun s creates a vertex v_s whose level equals the level of s plus one, and whose mass equals the sum of the masses of all the vertices in the solar system of s. Also, an *inter-level edge* between s and v_s is created and will be used in the placement phase. In the next superstep, every sun s adds an edge between v_s and v_t, if t is a sun of a solar system for which there are $k > 0$ inter-system links. The edge (v_s, v_t) is equipped with a *weight* equal to the maximum number of vertices involved in any of the k links. Finally, all vertices (except the newly created ones) deactivate themselves.

3.3 Placement Phase: DISTRIBUTED SOLAR PLACER

We now describe a DISTRIBUTED SOLAR PLACER algorithm, which behaves similarly to the SOLAR PLACER of FM3. After the coarsening phase, the only active vertices are those of the coarsest graph G_k. For this graph, the placement phase is not executed, and the computation goes directly to the single-level layout phase (described in the next subsection). The output of the single-level layout phase is an assignment of coordinates to all vertices of G_k. Then, the placement phase starts and its execution is as follows.

In the first superstep, every vertex broadcasts its coordinates. In the second superstep, all vertices whose level is one less than the level of the currently active vertices activate themselves, and hence will start receiving messages from the next superstep. In the same superstep, every vertex v forwards the received messages to the corresponding vertex v^* of lower level through its inter-level edge. Then v deletes itself. At the next superstep, if a vertex s receives a message, then s is the sun of a solar system. Thanks to the received messages, s becomes aware of the position of all suns of its neighboring solar systems. Hence, s exploits this information (and the data structure containing information on the inter-system links), to compute the coordinates of all planets and moons in its solar system, as for the SOLAR PLACER. Once this is done, s sends to every planet u of its solar system the coordinates of u. The coordinates of the moons are delivered through two-hop messages (that is, sent to planets and then forwarded).

3.4 Single-Level Layout Phase: The GILA Algorithm

This phase is based on the GILA algorithm, the distributed single-level force-directed algorithm described in Sect. 2. Recall that the execution of GILA is based on a set of parameters, whose tuning affects the trade-off between quality of the drawing and speed of the computation. The most important parameter is the maximum graph theoretic distance k between pairs of vertices for which the pairwise repulsive forces are computed. Also, there are further parameters that affect the maximum displacement of a vertex, at a given iteration of the

algorithm. The idea is to tune these parameters in order to achieve better quality for the coarser graphs, and shorter running times for the graphs whose size is closer to the original graph. Here we only describe how the parameter k has been experimentally tuned, since it is the parameter that mostly affect the trade-off between quality and running time. The other parameters have been set similarly. For the drawing of every graph G_i, the value of k is 6 if the number of edges m_i of G_i is below 10^3, it is 5 if $10^3 \leq m_i < 5 \cdot 10^3$, it is 4 if $5 \cdot 10^3 \leq m_i < 10^4$, it is 3 if $10^4 \leq m_i < 10^5$, it is 2 if $10^5 \leq m_i < 10^6$, and it is 1 if $m_i \geq 10^6$.

4 Experimental Analysis

We executed an experimental analysis whose objective is to evaluate the performance of Multi-GiLA. We aim to investigate both the quality of the produced drawings and the running time of the algorithm, also in terms of scalability when we increase the number of machines. We expect that Multi-GiLA computes drawings whose quality is comparable to that achieved by centralized multilevel force-directed algorithms. This is because the locality-based approximation scheme adopted by GiLA (used in the single-level layout phase) should be mitigated by the use of a graph hierarchy. Also, we expect Multi-GiLA to be able to handle graphs with several million edges in tens of minutes on an inexpensive PaaS infrastructure. Clearly, the use of a scalable vertex-centric distributed framework adds some unavoidable overhead, which may make Multi-GiLA not suited for graphs whose size is limited to a few hundred thousand of edges. Our experimental analysis is based on three benchmarks called RegularGraphs, RealGraphs, and BigGraphs, described in the following.

The RegularGraphs benchmark is the same used by Bartel et al. [8] in an experimental evaluation of various implementations of the three main phases of a multilevel force-directed algorithm (coarsening, placement, and single-level layout). It contains 43 graphs with a number of edges between 78 and 48,232, and it includes both real-world and generated instances [2]. See also Table 1 for more details. We used this benchmark to evaluate Multi-GiLA in terms of quality of the computed drawings. Since the coarsening phase plays an important role in the computation of a good drawing, we first evaluated the performance of our Distributed Solar Merger in terms of number of produced levels compared to the number of levels produced by the Solar Merger of FM3. It may be worth remarking that, in the experimental evaluation conducted by Bartel et al. [8], the Solar Merger algorithm showed the best performance in terms of drawing quality when used for the coarsening phase. Our experiments show that the number of levels produced by the two algorithms is comparable and follows a similar trend throughout the series of graphs. The Distributed Solar Merger produces one or two levels less than the Solar Merger in most of the cases, and this is probably due to some slight difference in the tuning of the two algorithms. To capture the quality of the computed drawings, we compared FM3 (the implementation available in the OGDF library [10]) and Multi-GiLA in terms of average number of crossings per edge (CRE), and normalized edge

Table 1. RegularGraphs: number of vertices (n), number of edges (m), average number of crossings per edge (CRE), normalized edge length std deviation (NELD).

			FM3		Multi-GiLA					FM3		Multi-GiLA	
Name	n	m	CRE	NELD	CRE	NELD	Name	n	m	CRE	NELD	CRE	NELD
karateclub	34	78	1.10	0.25	1.09	0.33	Grid_40_40_df	1,597	3,120	0.19	0.23	0.20	0.33
snowflake_A	98	97	0.00	0.25	0.11	0.21	Grid_40_40_sf	1,599	3,120	0.39	0.18	0.38	0.31
spider_A	100	160	3.06	0.24	2.86	0.27	ug_380	1,104	3,231	25.68	0.64	13.47	0.96
cylinder_010	97	178	0.35	0.16	0.72	0.08	esslingen	2,075	5,530	19.89	0.41	34.18	0.53
sierpinski_04	123	243	0.00	0.25	0.00	0.22	uk	4,824	6,837	0.07	0.36	0.06	0.65
tree_06_03	259	258	0.40	0.29	1.54	0.17	4970	4,970	7,400	0.01	0.23	0.01	0.46
rna	363	468	0.04	0.24	0.06	0.50	add20	2,395	7,462	60.38	0.50	100.44	0.50
protein_part	417	511	1.20	0.33	1.73	0.50	dg_1087	7,602	7,601	0.06	0.34	0.00	1.04
516	516	729	0.09	0.13	0.18	0.44	tree_06_05	9,331	9,330	8.63	0.47	19.65	0.93
Grid_20_20	400	760	0.00	0.13	0.00	0.23	add32	4,960	9,462	1.31	0.88	0.97	1.66
Grid_20_20_df	397	760	0.24	0.23	0.20	0.34	snowflake_C	9,701	9,700	0.00	0.64	0.00	0.40
Grid_20_20_sf	397	760	0.41	0.17	0.41	0.26	flower_005	930	13,521	48.76	0.61	45.24	0.61
dg_617_part	341	797	10.57	0.30	16.61	0.36	3elt	4,720	13,722	0.40	0.35	0.27	0.60
snowflake_B	971	970	0.00	0.42	0.00	0.39	data	2,851	15,093	2.15	0.39	2.52	0.64
tree_06_04	1,555	1,554	8.53	0.35	7.04	0.19	grid400_20	8,000	15,580	0.02	0.22	0.24	0.89
spider_B	1,000	1,600	7.03	0.24	8.26	0.73	spider_C	10,000	16,000	171.31	0.32	262.09	0.93
grid_rnd_032	985	1,834	0.00	0.15	0.00	0.30	grid_rnd_100	9,499	17,849	0.00	0.16	0.00	0.34
cylinder_032	985	1,866	0.46	0.19	0.44	0.39	sierpinski_08	9,843	19,683	0.09	0.44	0.03	0.70
cylinder_100	985	1,866	4.60	0.18	4.48	0.45	crack	10,240	30,380	0.00	0.26	0.00	0.42
sierpinski_06	1,095	2,187	0.06	0.34	0.03	0.63	4elt	15,607	45,878	0.52	0.39	0.30	0.62
flower_001	210	3,057	47.37	0.67	45.97	0.47	cti	16,840	48,232	10.19	0.39	10.26	0.71
Grid_40_40	1,600	3,120	0.00	0.15	0.00	0.32							

length standard deviation (NELD). The values of NELD are obtained by dividing the edge length standard deviation by the average edge length of each drawing. We chose FM3 for this comparison for two main reasons: (*i*) Multi-GiLA is partially based on distributed implementations of the Solar Merger and of the Solar Placer algorithms; (*ii*) FM3 showed the best trade-off between running time and quality of the produced drawings in the experiments of Hachul and Jünger [19]. The results of our experiments are reported in Table 1. The performance of Multi-GiLA is very close to that of FM3 in terms of CRE. In several cases Multi-GiLA produces drawings with a smaller value of CRE than FM3 (see, e.g., `ug_380`). Concerning the NELD, Multi-GiLA most of the times generates drawings with larger values than FM3. This may depend on how the length of the edges is set by the Distributed Solar Placer algorithm. However, also in this case the values of NELD follow a similar trend throughout the series of graphs. Figure 2 shows a visual comparison for some of the graphs. Similarly to FM3, Multi-GiLA is able to unfold graphs with a very regular structure and large diameter.

The RealGraphs and BigGraphs sets contain much bigger graphs than RegularGraphs, and are used to evaluate the running time of Multi-GiLA, especially in terms of strong scalability (i.e., how the running time varies on a given instance when we increase the number of machines). The RealGraphs set is composed of the 5 largest real-world graphs (mainly scale-free graphs) used in the experimental study of GiLA [5]. These graphs are taken from the Stanford Large Networks Dataset Collection [3] and from the Network Data

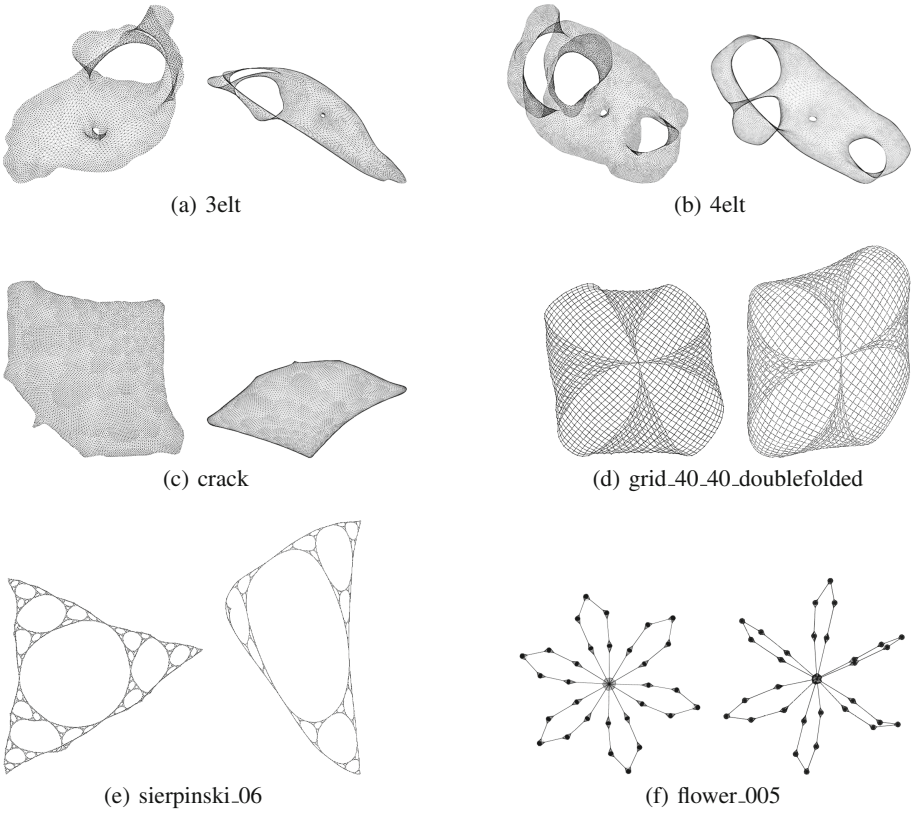


Fig. 2. Layouts of some RegularGraphs instances. For each graph, the drawing computed by FM3 (Multi-GiLA) is on the left (right).

Repository [4], and their number of edges is between 121,523 and 1,541,514. The BigGraphs set consists of 3 very large graphs with up to 12 million edges, taken from the collection of graphs described in [28][1]. Details about the RealGraphs and BigGraphs sets are in Table 2.

Table 2. Left: Details for RealGraphs. Right: Details for BigGraphs benchmark. Isolated vertices, self-loops, and parallel edges have been removed from the original graphs. The graphs are ordered by increasing number of edges.

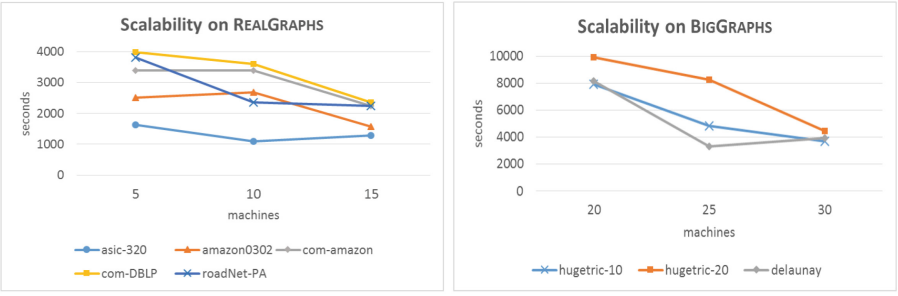
Name	n	m	Description	Name	n	m	Description
asic-320	121,523	515,300	circuit sim. problem	hugetric-10	6,600,000	10,000,000	Mesh
amazon0302	262,111	899,792	co-purchasing network	hugetric-20	7,100,000	10,700,000	Mesh
com-amazon	334,863	925,872	co-purchasing network	delaunay_n22	4,100,000	12,200,000	Triangulation
com-DBLP	317,080	1,049,866	collaboration network				
roadNet-PA	1,087,562	1,541,514	road network				

[1] See also http://www.networkrepository.com/.

Table 3 reports the running times of Multi-GiLA on the RealGraphs and BigGraphs instances, using increasing clusters of Amazon. Namely, for the RealGraphs instances, 5 machines were always sufficient to compute a drawing in a reasonable time, and using 15 machines the time is reduced by 35% on average. For the BigGraphs instances we used a number of machines from 20 to 30, and the reduction of the time going from the smallest to the largest cluster is even more evident than for the RealGraphs set (50% on average). Figure 3 depicts the trend of the data in Table 3, showing the strong scalability of Multi-GiLA. Figure 4 shows some layouts of RealGraphs and BigGraphs instances computed by Multi-GiLA and visualized (rendered) with LaGo. It is worth observing that some centralized algorithm may be able to draw quicker than Multi-GiLA graphs of similar size as those in the RealGraphs set (see e.g. [16]). This is partially justified by the use of a distributed framework such as Giraph, which introduces overheads in the computation that are significant for graphs of this size. However, this kind of overhead is amortized when scaling to larger graphs as those in the BigGraphs set. Also, using an optimized cluster rather than a PaaS infrastructure may improve the performance of the algorithm.

Table 3. Running time of Multi-GiLA on the RealGraphs and BigGraphs instances, using increasing clusters of Amazon.

	Running time (s)				Running time (s)		
Name	5 machines	10 machines	15 machines	Name	20 machines	25 machines	30 machines
asic-320	1,626	1,102	1,281	hugetric-10	7,923	4,828	3,679
amazon0302	2,518	2,696	1,577	hugetric-20	9,891	8,243	4,445
com-amazon	3,400	3,395	2,242	delaunay_n22	8,160	3,301	3,932
com-DBLP	4,000	3,612	2,366				
roadNet-PA	3,813	2,369	2,241				



(a) (b)

Fig. 3. Scalability of Multi-GiLA on RealGraphs instances.

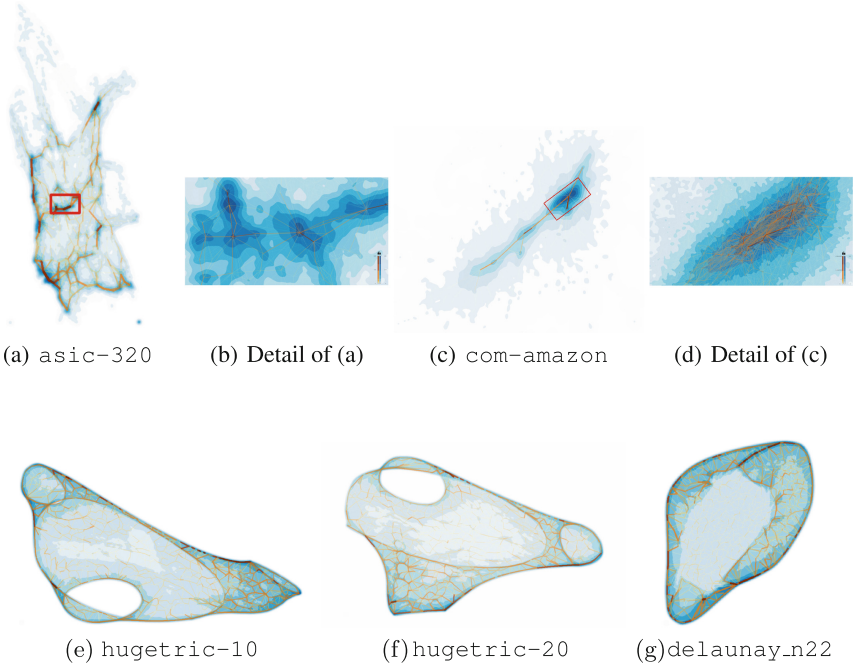


Fig. 4. Layouts of (a–d) RealGraphs instances and (e–f) BigGraphs instances computed by Multi-GiLA and visualized (rendered) with LaGo.

5 Conclusions and Future Research

As far as we know, Multi-GiLA is the first multilevel force-directed technique working in a distributed vertex-centric framework. Its communication protocol allows for an effective computation of a coarse graph hierarchy. Experiments indicate that the quality of the computed layouts compares with that of drawings computed by popular centralized multilevel algorithms and that it exhibits high scalability to very large graphs. Our source code is made available to promote research on the subject and to allow replicability of the experiments. In the near future we will investigate more coarsening techniques and single-level layout methods for a vertex-centric distributed environment.

References

1. http://www.geeksykings.eu/multigila/
2. http://ls11-www.cs.tu-dortmund.de/staff/klein/gdmult10
3. http://snap.stanford.edu/data/index.html
4. http://www.networkrepository.com/
5. Arleo, A., Didimo, W., Liotta, G., Montecchiani, F.: A distributed force-directed algorithm on Giraph: design and experiments. ArXiv e-prints (2016). http://arxiv.org/abs/1606.02162

6. Arleo, A., Didimo, W., Liotta, G., Montecchiani, F.: A distributed multilevel force-directed algorithm. ArXiv e-prints (2016). http://arxiv.org/abs/1608.08522
7. Arleo, A., Didimo, W., Liotta, G., Montecchiani, F.: A million edge drawing for a fistful of dollars. In: Di Giacomo, E., Lubiw, A. (eds.) GD 2015. LNCS, vol. 9411, pp. 44–51. Springer, Heidelberg (2015). doi:10.1007/978-3-319-27261-0_4
8. Bartel, G., Gutwenger, C., Klein, K., Mutzel, P.: An experimental evaluation of multilevel layout methods. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 80–91. Springer, Heidelberg (2011). doi:10.1007/978-3-642-18469-7_8
9. Chae, S., Majumder, A., Gopi, M.: Hd-graphviz: Highly distributed graph visualization on tiled displays. In: ICVGIP 2012, pp. 43: 1–43: 8. ACM (2012)
10. Chimani, M., Gutwenger, C., Jünger, M., Klau, G.W., Klein, K., Mutzel, P.: The open graph drawing framework (OGDF). In: Tamassia, R. (ed.) Handbook on Graph Drawing and Visualization, pp. 543–569. CRC, Boca Raton (2013). http://www.ogdf.net/
11. Ching, A.: Giraph: large-scale graph processing infrastructure on hadoop. In: Hadoop Summit (2011)
12. Ching, A., Edunov, S., Kabiljo, M., Logothetis, D., Muthukrishnan, S.: One trillion edges: graph processing at facebook-scale. PVLDB **8**(12), 1804–1815 (2015)
13. Didimo, W., Montecchiani, F.: Fast layout computation of clustered networks: algorithmic advances and experimental analysis. Inf. Sci. **260**, 185–199 (2014). http://dx.doi.org/10.1016/j.ins.2013.09.048
14. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Exp. **21**(11), 1129–1164 (1991)
15. Gajer, P., Goodrich, M.T., Kobourov, S.G.: A multi-dimensional approach to force-directed layouts of large graphs. Comput. Geom. **29**(1), 3–18 (2004)
16. Godiyal, A., Hoberock, J., Garland, M., Hart, J.C.: Rapid multipole graph drawing on the GPU. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 90–101. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00219-9_10
17. Hachul, S.: A potential field based multilevel algorithm for drawing large graphs. Ph.D. thesis, University of Cologne (2005). http://kups.ub.uni-koeln.de/volltexte/2005/1409/index.html
18. Hachul, S., Jünger, M.: Drawing large graphs with a potential-field-based multilevel algorithm. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 285–295. Springer, Heidelberg (2005). doi:10.1007/978-3-540-31843-9_29
19. Hachul, S., Jünger, M.: Large-graph layout algorithms at work: an experimental study. J. Graph Algorithms Appl. **11**(2), 345–369 (2007)
20. Hadany, R., Harel, D.: A multi-scale algorithm for drawing graphs nicely. Discrete Appl. Math. **113**(1), 3–21 (2001)
21. Harel, D., Koren, Y.: A fast multi-scale method for drawing large graphs. J. Graph Algorithms Appl. **6**(3), 179–202 (2002)
22. Hinge, A., Auber, D.: Distributed graph layout with Spark. In: IV 2015, pp. 271–276. IEEE (2015)
23. Hu, Y.: Efficient, high-quality force-directed graph drawing. Mathematica J. **10**(1), 37–71 (2005)
24. Ingram, S., Munzner, T., Olano, M.: Glimmer: Multilevel MDS on the GPU. IEEE Trans. Vis. Comput. Graph. **15**(2), 249–261 (2009)
25. Kobourov, S.G.: Force-directed drawing algorithms. In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization. CRC Press, Boca Raton (2013)
26. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: A system for large-scale graph processing. In: SIGMOD 2010, pp. 135–146. ACM (2010)

27. Mueller, C., Gregor, D., Lumsdaine, A.: Distributed force-directed graph layout and visualization. In: EGPGV 2006, pp. 83–90. Eurographics (2006)
28. Rossi, R.A., Ahmed, N.K.: An interactive data repository with visual analytics. SIGKDD Explor. **17**(2), 37–41 (2016). http://networkrepository.com
29. Sharma, P., Khurana, U., Shneiderman, B., Scharrenbroich, M., Locke, J.: Speeding up network layout and centrality measures for social computing goals. In: Salerno, J., Yang, S.J., Nau, D., Chai, S.-K. (eds.) SBP 2011. LNCS, vol. 6589, pp. 244–251. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19656-0_35
30. Tikhonova, A., Ma, K.: A scalable parallel force-directed graph layout algorithm. In: EGPGV 2008, pp. 25–32. Eurographics (2008)
31. Valiant, L.G.: A bridging model for parallel computation. Commun. ACM **33**(8), 103–111 (1990)
32. Vaquero, L.M., Cuadrado, F., Logothetis, D., Martella, C.: Adaptive partitioning for large-scale dynamic graphs. In: ICDCS 2014, pp. 144–153. IEEE (2014)
33. Walshaw, C.: A multilevel algorithm for force-directed graph-drawing. J. Graph Algorithms Appl. **7**(3), 253–285 (2003)
34. Yunis, E., Yokota, R., Ahmadia, A.: Scalable force directed graph layout algorithms using fast multipole methods. In: ISPDC 2012, pp. 180–187. IEEE (2012)
35. Zinsmaier, M., Brandes, U., Deussen, O., Strobelt, H.: Interactive level-of-detail rendering of large graphs. IEEE Trans. Vis. Comput. Graph. **18**(12), 2486–2495 (2012)