SGV: Spatial Graph Visualization*

Tarlan Bahadori
tbaha001@ucr.edu
UC Riverside
California, USA

Alvin Chiu
chiua13@uci.edu
UC Irvine
California, USA

Ahmed Eldawy
eldawy@ucr.edu
UC Riverside
California, USA

Michael T. Goodrich
goodrich@uci.edu
UC Irvine
California, USA

Abstract

Spatial graphs, where nodes carry geographic location information, are vital for modeling complex relationships in domains such as location-based social networks, transportation systems, and knowledge graphs. However, it is challenging to visualize large spatial graphs while simultaneously showing edge connections and vertex spatial fidelity, especially when the location of each vertex is imprecise. We present a distributed geospatial force-directed framework that visualizes spatial graphs where location can be represented as a point, multi-point, linestring, or polygon. It integrates three models for anchoring forces: centroidal, inside-out, and closest-point. The algorithm is formulated as relational operations and runs end-to-end on Apache Spark/SparkSQL, achieving near-linear scaling. Experiments on train networks, author–publication graphs, and location-based social networks show clearer layouts that balance edge lengths and spatial fidelity while reducing crossings.

CCS Concepts

• **Information systems** → **Geographic information systems**; • **Human-centered computing** → *Information visualization.*

Keywords

Geospatial Anchoring, Force-Directed Graph Layout, Spatial Graph

ACM Reference Format:
Tarlan Bahadori, Alvin Chiu, Ahmed Eldawy, and Michael T. Goodrich. 2025. SGV: Spatial Graph Visualization. In *The 33rd ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '25), November 3–6, 2025, Minneapolis, MN, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3748636.3762772

1 Introduction

Graphs are a flexible model for complex relationships in location-based social networks [4, 19], transportation systems, academic collaboration [27], and knowledge graphs [25]. In many of these settings, nodes carry spatial context that is uncertain or multi-valued, making it challenging to present a layout that respects geography while revealing graph structure [23]. Purely topological drawings can distort the map, whereas purely spatial placements often tangle edges and obscure patterns.

Force-directed layouts model vertices as mutually repelling particles and edges as springs, yielding clear, interpretable drawings[6, 8, 11, 12, 16, 24]. As noted by prior work [7, 13], naïve all-pairs repulsion is $O(n^2)$ and the repulsion energy is non-convex, so layouts can stall in cluttered local minima. Distributed variants partition computation or use big-data engines, but had high network overheads [14, 21]. In parallel, geospatial/anchored layouts balance map faithfulness and readability by fixing or softly constraining node positions [20]. Visual clutter can be reduced with edge bundling and geometry-based clustering [5, 15], though these do not themselves enforce geographic constraints. Region-based anchoring via centroid or boundary forces has been explored, typically for single-region constraints or polygon anchors at moderate scale [3, 26].

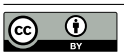
This paper presents a distributed spatial graph layout that fuses uncertain geography with force-directed structure at scale. We extend anchoring beyond polygons to multi-point sets and linestrings, pulling vertices toward multiple candidate locations without hard pinning. Each iteration is expressed in SparkSQL: an equi-join over edges for attraction, spatial joins with anchor geometries, and a spatial self-join for near-neighbor repulsion, followed by partition-local accumulation with periodic summary rebroadcast. This communication-aware design avoids all-pairs interactions, reduces shuffles, and achieves near-linear scaling while preserving geographic fidelity.

We evaluate on three geospatial graph case studies: transportation networks, author–publication graphs, and location-based social networks, showing cleaner pattern revelation than non-spatial layouts and up to 20× speed-ups on very large inputs. To our knowledge, no prior system simultaneously supports multi-anchor constraints (multi-point, linestring, polygon) and big-data scalability within a single force-directed framework.

2 Problem Definition

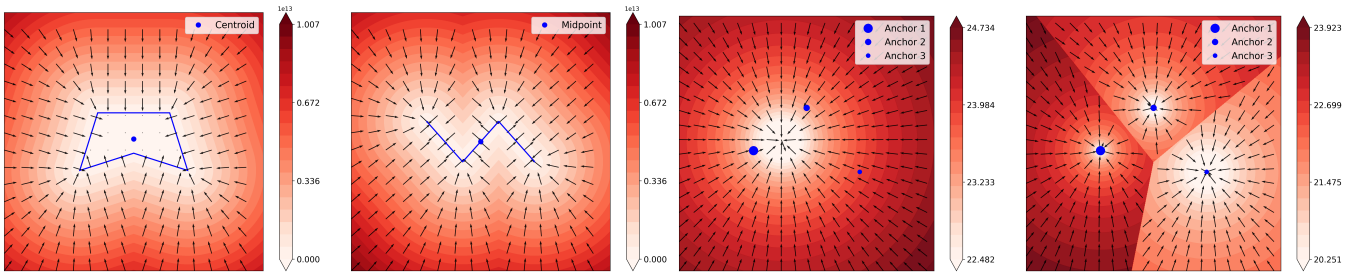
Rendering large spatial graphs at scale presents two competing challenges: *1) Preserve visual readability* by minimizing edge crossings and vertex overlaps, making edge lengths similar, and minimizing the overall used drawing area [13]. *2) Maintain spatial fidelity* by anchoring vertices to geographic regions. To address this, we model each dataset as a *geospatial undirected graph*, $G = (V, E)$, where each vertex $u \in V$ has a location $\ell_u \in \mathbb{R}^2$ and an anchor region $A_u \subseteq \mathbb{R}^2$ (point, multi-point set, polygon, or linestring). Our goal is to compute a *graph layout* that assigns a location $\mathbf{r}_u \in \mathbb{R}^2$ for each vertex u and maps edges to line segments joining their endpoints while satisfying: 1) **connectivity:** connected vertices are reasonably close to each other, 2) **edge homogeneity:** edge lengths are roughly of the same size, 3) **spatial fidelity:** vertices are close to their anchor location, and 4) **scalability:** the solution should scale to large graphs.

*This work is supported in-part by the National Science Foundation under grants 2046236, 2212129, and 2520163



SIGSPATIAL '25, Minneapolis, MN, USA

ACM ISBN 979-8-4007-2086-4/2025/11
https://doi.org/10.1145/3748636.3762772



(a) Closest-Point - Polygon **(b) Closest-Point - Linestring** **(c) Centroidal - Multi-point** **(d) Closest-Point - Multi-point**

Figure 1: Visualization of anchor force fields in different cases. Color indicates magnitude and vectors indicate direction.

Traditional force-directed graph visualization algorithms focus on the first two goals by modeling nodes as charged particles and edges as loaded springs. This keeps connected vertices close to each other while decluttering the layout. This paper enriches traditional algorithms by integrating anchoring forces that satisfy spatial fidelity. In addition, our algorithm is designed to work on the distributed Spark framework that satisfies the scalability goal.

Extended Anchoring Metrics. Let each vertex u have position $\mathbf{r}_u$ and anchor region A_u, with center of mass c_u and the closest point on anchor region to $\mathbf{r}_u$ is denoted by p_u. We define the *size* of A_u as

$$\mu(A_u) = \begin{cases} 1, & A_u \text{ is a point,} \\ |A_u|, & A_u \text{ is a finite set of points,} \\ \text{Length}(A_u), & A_u \text{ is a linestring,} \\ \text{Area}(A_u), & A_u \text{ is a polygon,} \end{cases}$$

and let

$$\mathbf{c}_u = \frac{1}{\mu(A_u)} \int_{A_u} \mathbf{x}\, d\mu(\mathbf{x}), \quad \mathbf{p}_u = \arg\min_{\mathbf{y} \in A_u} \|\mathbf{r}_u - \mathbf{y}\|.$$

In practice:

- *Point anchors:* $\mathbf{c}_u = \mathbf{p}_u = A_u$.
- *Multi-point anchors:* $\mathbf{c}_u$ is the arithmetic weighted mean of the points; $\mathbf{p}_u$ is the nearest point in the set to r_u.
- *Linestring anchors:* $\mathbf{c}_u$ is the midpoint on the linestring; $\mathbf{p}_u$ is the projection of r_u on the linestring.
- *Polygon anchors:* $\mathbf{c}_u$ is the area centroid; $\mathbf{p}_u$ is the closest boundary point of the polygon to r_u.

Using an anchoring force coefficient α, we define three force variants that are based on Hooke's Law for spring force:

Centroidal Anchor Force.

$$\mathbf{F}_u^{\text{anchor}} = \alpha\,(\mathbf{c}_u - \mathbf{r}_u).$$

Inside–Out Anchor Force.

$$\mathbf{F}_u^{\text{anchor}} = \begin{cases} \alpha\,(\mathbf{c}_u - \mathbf{r}_u), & \mathbf{r}_u \notin A_u, \\ \mathbf{0}, & \mathbf{r}_u \in A_u. \end{cases}$$

Closest–Point Anchor Force.

$$\mathbf{F}_u^{\text{anchor}} = \begin{cases} \alpha\,(\mathbf{p}_u - \mathbf{r}_u), & \mathbf{r}_u \notin A_u, \\ \mathbf{0}, & \mathbf{r}_u \in A_u. \end{cases}$$

Anchoring forces are vectors: they have magnitude and a direction from the vertex position to the target (centroid or closest point).

Figure 1(a–d) shows the force fields for four cases; darker color means larger magnitude and arrows indicate direction. (a) Polygon anchor + closest point model: smooth field toward the nearest boundary. (b) Linestring anchor + closest point model: similarly smooth toward the curve. (c) Multi-point anchor + centroidal: smooth pull toward the centroid. (d) Multi-point anchor + closest point model: Voronoi-like regions with discontinuities.

3 Relational Query Modeling with SparkSQL

We express each force component as a relational query over the base relations `Vertices(vid, currx, curry, anchor)` and `Edges(srcid, dstid)`. The primary key of `Vertices` is `vid`; `Edges` has composite key `(srcid,dstid)`. The `anchor` column in Vertices holds the anchoring geometry (point, multipoint, linestring, or polygon). If the anchor spans the whole plane, the vertex is effectively free. Attractive forces are computed via equi-joins between `Edges` and `Vertices`; pairwise repulsion is, in principle, a Cartesian product over `Vertices` but we approximate it via a distance-limited spatial self-join for scalability.

Each iteration sums attractive, repulsive, and anchoring contributions to obtain (F_x, F_y) per vertex, then advances positions.

Attractive Forces (Edges ⋈ Vertices).

Attractive (spring) forces. Following the spring–embedder model (not strictly physical [13]), each edge pulls its endpoints toward an ideal length L (spring rest length). For edge (u, v) with positions $\mathbf{p}_u, \mathbf{p}_v$, let $\Delta\mathbf{r}_{uv} = \mathbf{p}_u - \mathbf{p}_v$ and clamp small separations by $d = \max(\|\Delta\mathbf{r}_{uv}\|, \varepsilon)$. The attractive force on u due to v is

$$\mathbf{F}_{u \leftarrow v}^{\text{att}} = -\frac{d}{L}\,\Delta\mathbf{r}_{uv},$$

with $\mathbf{F}_{v \leftarrow u}^{\text{att}} = -\mathbf{F}_{u \leftarrow v}^{\text{att}}$ by symmetry; the net attractive force on vertex i sums contributions over its incident edges.

```
WITH Endpts AS (
  SELECT e.srcid AS u, e.dstid AS v,
    us.currx AS ux, us.curry AS uy, vs.currx AS vx, vs.curry AS vy,
    (vs.currx - us.currx) AS dx, (vs.curry - us.curry) AS dy,
    (GREATEST(SQRT(POW(vs.currx-us.currx,2) +
        POW(vs.curry-us.curry,2)), :eps) / :L) AS coef
  FROM Edges e
  JOIN Vertices us ON e.srcid = us.vid
  JOIN Vertices vs ON e.dstid = vs.vid
),
```

```
PairForces AS (
  SELECT u AS vid,  coef * dx AS fx,  coef * dy AS fy FROM Endpts
  UNION ALL
  SELECT v AS vid, -coef * dx AS fx, -coef * dy AS fy FROM Endpts
)
SELECT vid, SUM(fx) AS fax, SUM(fy) AS fay
FROM PairForces
GROUP BY vid;
```

Repulsive Forces (Spatial Self-Join). For distinct vertices $u \neq v$, let $\mathbf{r}_{uv} = \mathbf{p}_u - \mathbf{p}_v$. The repulsive force $\mathbf{F}^{\text{rep}}$ is defined as

$$d^2 \;=\; \max\!\big(\|\mathbf{r}_{uv}\|^2,\; \varepsilon^2\big), \qquad \mathbf{F}^{\text{rep}}_{u \leftarrow v} \;=\; c_{\text{rep}}\,\frac{\mathbf{r}_{uv}}{d^2}.$$

where c_{rep} is a constant. Exact evaluation is $O(|V|^2)$, so we approximate by a spatial self-join that keeps only neighbors within cutoff radius r. The net repulsion on vertex i is:

$$\mathbf{F}^{\text{rep}}_i \;=\; \sum_{j \,\in\, \mathcal{N}_r(i)} c_{\text{rep}}\,\frac{\mathbf{p}_i - \mathbf{p}_j}{\max(\|\mathbf{p}_i - \mathbf{p}_j\|^2, \varepsilon^2)}.$$

```
WITH Pairs AS (
  SELECT a.vid  AS u, b.vid  AS v,
    (a.currx - b.currx) AS dx, (a.curry - b.curry) AS dy,
    POW(a.currx - b.currx, 2) + POW(a.curry - b.curry, 2) AS d2
  FROM Vertices a JOIN Vertices b ON a.vid <> b.vid
  WHERE POW(a.currx - b.currx, 2) + POW(a.curry - b.curry, 2) < :r2
),
Rep AS (
  SELECT
    u AS vid,
    :crep * dx / GREATEST(d2, :eps2) AS frx,    -- :eps2 = ε^2
    :crep * dy / GREATEST(d2, :eps2) AS fry
  FROM Pairs
)
SELECT vid, SUM(frx) AS frx, SUM(fry) AS fry
FROM Rep
GROUP BY vid;
```

Anchoring forces. Each vertex v stores an anchor geometry $A(v)$ (WKT in `Vertices.anchor`) and a chosen rule where m is one of centroidal, closest, or inside–out. Let $\mathbf{p}_v = (x_v, y_v)$. We pull v toward a rule-specific target $\boldsymbol{\tau}_m(v)$: centroidal uses centroid$(A(v))$; closest uses the closest point on $A(v)$ when $\mathbf{p}_v \notin A(v)$ (no pull if inside); inside–out uses the closest point on $\partial A(v)$ when $\mathbf{p}_v \in A(v)$ (no pull if outside). The force is

$$\mathbf{F}^{\text{anc}}(v) = k_n\big(\boldsymbol{\tau}_m(v) - \mathbf{p}_v\big).$$

Implementation. In SparkSQL we compute $\boldsymbol{\tau}_m(v)$ using Sedona/JTS UDFs (e.g., `ST_Contains`, `ST_Centroid`, `ST_ClosestPoint`) and emit per-vertex force components `Anc(vid, fnx, fny)`.

```
-- Minimal pattern: delegate geometry details to UDFs
-- :method one of {'centroidal','closest','insideout'}
WITH Anc AS (
  SELECT v.vid,
    :kn * (ANCHOR_TX(v.anchor, v.currx, v.curry, :method) - v.currx)
      AS fnx,
    :kn * (ANCHOR_TY(v.anchor, v.currx, v.curry, :method) - v.curry)
      AS fny
  FROM Vertices v
)
SELECT * FROM Anc;
```

Notes. The UDFs encapsulate the simple predicates (e.g., "if outside, pull to closest point; if inside and inside-out, pull to boundary; if centroidal, pull to centroid).

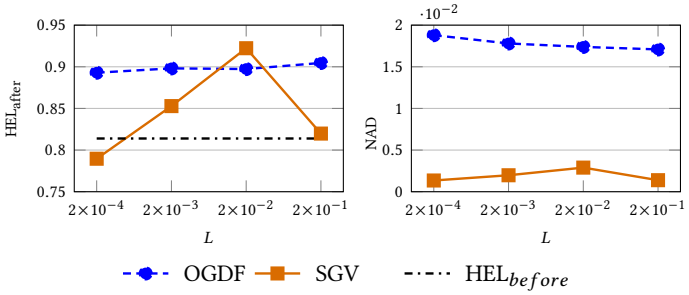


Figure 2: Author–publication network: HEL (left) and NAD (right) vs. L (ideal edge length).

Position Update and Iteration. Let the net force on vertex i be $\mathbf{F}_i = \mathbf{F}^{\text{att}}_i + \mathbf{F}^{\text{rep}}_i + \mathbf{F}^{\text{anc}}_i$. A per-iteration temperature τ_t caps the maximum movement. The raw displacement equals the net force and is clipped to length τ_t before applying. Relationally, we materialize three per-vertex force tables—`Attr`, `Rep`, and `Anc`—take a single bag union (`UNION ALL`), and aggregate by `vid` to obtain `Total` with fields (vid, F_x, F_y). We left-join `Total` onto `Vertices` and update with a temperature cap: $\Delta x = \text{clip}(F_x, \tau_t)$, $\Delta y = \text{clip}(F_y, \tau_t)$, newx = currx + Δx, newy = curry + Δy.

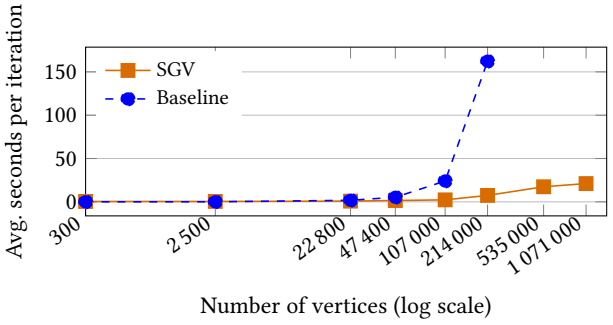


Figure 3: Per-iteration runtime on Gowalla graphs: SGV vs. OGDF across increasing vertex counts (log-scaled x-axis).

4 Experimental Evaluation

Setup. We ran experiments on a 12-node Spark cluster. The master has 2×8-core Intel Xeon E5-2609 v4 (1.70 GHz) and 128 GB RAM; each worker has 2×6-core E5-2603 v4 (1.70 GHz) and 64 GB RAM. All nodes run CentOS 7.5; SGV code is available on GitHub [1].

Datasets and scalability protocol. We use three graph families: rail networks, author–publication, and location-based social networks (LBSN), to illustrate layout quality and scalability. All scalability experiments use a Gowalla-derived dataset [17] constructed at multiple sizes via: (i) **BFS subsets**: starting from random seeds, we extract 2–3-hop neighborhoods to obtain smaller graphs while preserving local structure; and (ii) $k\times$ **augmentations**: we fit a 2D density to the original vertex locations and a degree histogram, then sample $k|V|$ new coordinates and desired degrees; edges are formed by distance-biased stub matching, preserving spatial clustering and degree distribution at larger scales.

Evaluation Metrics.

Homogeneous Edge Length (HEL). HEL [18] measures edge-length uniformity. Let E be the edge set, $m = |E|$, $\ell_j = \|e_j\|$, $\bar{\ell}$ the mean edge length, and $\ell_{\max}$ the maximum. Define

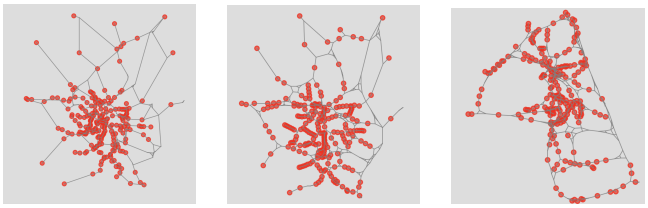
$$\mathrm{HEL} = 1 - \frac{1}{m}\sum_{j=1}^{m}\left|\frac{\ell_j - \bar{\ell}}{\max(\bar{\ell},\ \ell_{\max} - \bar{\ell})}\right|.$$

HEL lies in $[0, 1]$ (1 = perfectly equal lengths; higher is better). We compute ℓ_j as geodesic (haversine) distances from vertex lat/lon and report HEL before/after SGV to quantify layout regularity.

Normalized Anchor Distance (NAD). For vertex u at position $\mathbf{r}_u$ and anchor geometry $A(u) \subset \mathbb{R}^2$, define

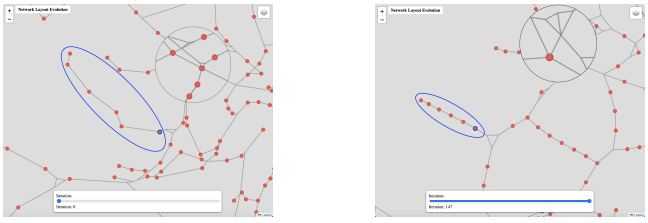
$$d_A(u) = \min_{p \in A(u)} \|\mathbf{r}_u - p\|, \qquad \widetilde{d}_A = \frac{1}{|V|\, D_{\mathrm{MBR}}} \sum_{u \in V} d_A(u),$$

where D_{MBR} is the diagonal length of the dataset's minimum bounding rectangle. Variants: for *centroidal* anchoring, use $d_A(u) = \|\mathbf{r}_u - \mathrm{centroid}(A(u))\|$; for *inside–out*, set $d_A(u) = 0$ if $\mathbf{r}_u \in A(u)$ and otherwise use the centroid distance.



(a) Initial layout **(b) Final w/ anchors** **(c) Final w/o anchors**

Figure 4: Railroad network in Paris: (a) before and (b) after applying SGV; (c) result with no geospatial anchoring forces.



(a) Initial Layout **(b) After SGV Convergence**

Figure 5: Railroad network in Paris (zoomed): (a) Initial; (b) After SGV. The blue station anchor pulls its branch into the main network, yielding more uniform edge lengths and spacing; the black circle shows reduced clutter and clearer routes.

Results & Analysis. Figure 3 shows the effect of increasing number of vertices (and edges) in the dataset on the runtime when using SGV compared to the baseline based on Fruchterman-Reingold implementation from the Open Graph Drawing Framework [2]. The baseline could not handle the two last augmented datasets.

Author–publication graphs are bipartite, with disjoint author and paper vertex sets and edges for authorship. Using OpenAlex [22], we collected ego networks (an author and their k-hop co-authors and papers). As shown in Fig. 2, anchoring (SGV) versus non-anchored Fruchterman–Reingold yields a clear trade-off: SGV lowers NAD (higher spatial fidelity) while keeping HEL broadly comparable (occasionally a bit lower or higher, depending on L).

Spatial rail/metro maps are a compelling use case for SGV. Using the Europe railroad lines [9] and stations [10], we simplify tracks, drop degree-2 junctions that are not stations, and retain junctions of degree ≥ 3 to preserve topology. Each station is a vertex anchored to a $0.002° \times 0.002°$ square around its geocoordinate; major junction stations receive stronger anchoring, while topology-only junction points remain unanchored. Figure 4 shows the resulting layout: without anchoring the geographic outline is lost and the topology tangles (Fig. 4c); with appropriate anchoring (Fig. 4b) the continental shape and readability improve, and a zoomed comparison further highlights decluttering and clearer routes (Fig. 5).

References

[1] Tarlan Bahadori. 2025. SGV Source Code. https://github.com/tarlaun/fdgv.
[2] Markus Chimani et al. 2014. The Open Graph Drawing Framework (OGDF). In *Handbook of Graph Drawing and Visualization*. CRC Press, Chapter 17.
[3] Alvin Chiu et al. 2024. Polygonally Anchored Graph Drawing (Extended Abstract). In *32nd International Symposium on Graph Drawing and Network Visualization*.
[4] Eunjoon Cho et al. 2011. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD*.
[5] Weiwei Cui et al. 2008. Geometry-Based Edge Clustering for Graph Visualization. *IEEE Transactions on Visualization and Computer Graphics* (2008).
[6] Giuseppe Di Battista et al. 1999. *Graph Drawing*.
[7] Tim Dwyer, Ken Marriott, and Michael Wybrow. 2006. Integrating Edge Routing into Force-Directed Layout. In *Graph Drawing*.
[8] Peter Eades. 1984. A Heuristic for Graph Drawing. *Congressus Numerantium* 42.
[9] EuroGeographics. 2016. Europe – Rail Road. https://public.opendatasoft.com/explore/dataset/europe-rail-road/information/ Accessed: 2025-05-05.
[10] EuroGeographics. 2016. Europe – Railway Station. https://public.opendatasoft.com/explore/dataset/europe-railway-station/export/ Accessed: 2025-05-05.
[11] Thomas M. J. Fruchterman and Edward M. Reingold. 1991. Graph Drawing by Force-Directed Placement. *Software: Practice and Experience* 21, 11 (1991).
[12] Pawel Gajer et al. 2004. A multi-dimensional approach to force-directed layouts of large graphs. *Computational Geometry* (2004).
[13] Stefan Hachul. 2005. *A Potential-Field-Based Multilevel Algorithm for Drawing Large Graphs*. Ph. D. Dissertation. Universität zu Köln.
[14] Antoine Hinge et al. 2015. Distributed Graph Layout with Spark. In *Proceedings of the 2015 19th International Conference on Information Visualisation (IV '15)*.
[15] Danny Holten and Jarke J. van Wijk. 2009. Force-Directed Edge Bundling for Graph Visualization. *Computer Graphics Forum* 28, 3 (2009), 983–990.
[16] Tomihisa Kamada and Satoru Kawai. 1989. An Algorithm for Drawing General Undirected Graphs. *Inform. Process. Lett.* 31 (1989).
[17] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.
[18] Giordano Da Lozzo et al. 2015. Drawing georeferenced graphs-combining graph drawing and geographic data. In *IVAPP*.
[19] Wei Luo et al. 2011. Spatial-social network visualization for exploratory data analysis. In *Proc. 3rd ACM SIGSPATIAL LBSN*.
[20] Kelly A. Lyons, Henk Meijer, and David Rappaport. 1998. Algorithms for Cluster Busting in Anchored Graph Drawing. *J. Graph Algorithms Appl.* (1998).
[21] Christopher Mueller et al. 2006. Distributed Force-Directed Graph Layout and Visualization.. In *EGPGV@ EuroVis/EGVE*.
[22] Jason Priem, Heather Piwowar, and Richard Orr. 2022. OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts.
[23] Sarah Schöttler et al. 2021. Visualizing and Interacting with Geospatial Networks: A Survey and Design Space. *Comput. Graph. Forum* 40, 6 (2021).
[24] Roberto Tamassia. 2013. *Handbook of Graph Drawing and Visualization*.
[25] Sizhe Wang et al. 2023. GeoGraphViz: Geographically constrained 3D force-directed graph for knowledge graph visualization. *Transactions in GIS* (2023).
[26] Hsiang-Yun Wu et al. 2022. Multi-level Area Balancing of Clustered Graphs. *IEEE Trans. Visualization and Computer Graphics* 28, 7 (2022), 2682–2696.
[27] Ye Yu et al. 2022. NcoVis: A Visual Analysis Framework for Exploring Academic Collaboration Networks under New Collaborative Relationships. In *IEEE CSCWD*.