Force-Directed Graph Layouts by Edge Sampling

Robert Gove*

Two Six Labs

ABSTRACT

Recent work shows that sampling algorithms can be an effective tool for graph visualization. This paper extends prior work by applying edge sampling algorithms to speed up the spring force calculation in force-directed graph layout algorithms. An experiment on 72 graphs finds that some sampling algorithms achieve comparable quality as no sampling. This result is confirmed with visualizations of the graph layout results. However, runtime improvements are small, especially for graphs with 10,000 vertices or fewer, indicating that the runtime savings might not be worth the risk to layout quality. Therefore, this paper suggests that accurate spring forces may be more important to force-directed graph layout algorithms than accurate electric forces. A copy of this paper plus the code and data to reproduce the results are available at <https://osf.io/4ja29/>

Index Terms: Human-centered computing—Visualization—Visualization techniques—Graph drawings; Theory of computation—Design and analysis of algorithms—Streaming, sublinear and near linear time algorithms—Sketching and sampling

1 INTRODUCTION

Spring-electric algorithms are one of the most common types of force-directed layout algorithms, with benefits such as good quality layouts [8, 21, 27, 29] and conceptual simplicity [2, 11, 23, 25]. Despite recent advances, running these algorithms on a large graph can take minutes or hours to finish. For a graph $G = (V, E)$, the fastest spring-electric algorithm runs in $O(|E|)$ time to compute the spring (edge) forces, and $O(|V|)$ time to compute the electric (vertex) forces [16]. However, $|E| = O(|V|^2)$ for dense graphs, so the worst case running time is $O(|V|^2)$ due to the spring force calculations.

To improve the runtime of spring-electric force-directed graph layout algorithms, this paper’s first contribution is a family of edge sampling algorithms for calculating spring forces. This extends recent work on vertex sampling, which has been shown to be an effective technique to speed up electric force calculations [16]. The edge sampling algorithms either (1) sample a fixed percentage of edges, or (2) sample edges following a geometric series.

This paper’s second contribution is an analysis of several experiments on these edge sampling algorithms using 72 graphs. Analysis suggests that spring force calculations are a small percent of the overall layout algorithm runtime. When edges are sampled using a large fixed percentage (50%), the quality tends to be close to the standard spring force calculation algorithm (i.e. no sampling), but runtime improvements are small. When the sample size is smaller or follows a geometric series the layout quality tends to be worse and probably is not of practical value. This negative result indicates that researchers and practitioners may wish to avoid edge sampling for force-directed graph layout algorithms. When these results are taken in context of the success of electric force approximation algorithms, this indicates that accurate spring forces may be more important for graph layout quality than accurate electric forces.

2 BACKGROUND

Spring-electric algorithms are a type of force-directed graph layout algorithm that model edges as springs and vertices as charged particles. The algorithms run an iterative physics simulation to compute vertex positions. At each iteration, the brute force algorithm runs in $O(|V|^2)$ time to compute the pairwise electric forces, and $O(|E|)$ time to compute the spring forces.

There are many proposals to improve the runtime of the electric forces, such as hierarchical aggregations [3, 5, 18] and random vertex sampling [16], which reduced the electric force runtime to $O(|V|)$. However, we are not aware of any proposals to reduce the spring force calculation runtime. Speeding up the spring force calculation can potentially have a big impact, since $O(|E|) = O(|V|^2)$ for dense graphs. The closest known related work is multi-scale graph layout algorithms [1, 19, 20], which use graph filtration to collapse vertices and edges to produce a hierarchy of coarsened versions of the graph. The layout algorithms then typically run a force-directed layout algorithm on the coarsened graphs, and use the resulting vertex positions to derive initial positions for the less coarse graphs. Because these multi-scale algorithms also use force-directed layout algorithms, a method for speeding up spring force calculations would also benefit multi-scale layout algorithms.

Visualizations of moderate to large relational data sets tend to be slow to calculate or render, but researchers have shown that sampling can be an effective technique for speeding up these visualizations. Successful applications include speeding up multi-dimensional scaling [6, 26] and speeding up electric force calculations for spring-electric force-directed layouts [16]. Of particular note is the application of random edge sampling to produce proxy graphs. (Proxy graphs are small representative versions of larger graphs that can be used to speed up rendering.) For proxy graphs, random edge sampling performs as well as spectral sparsification [10], a more complicated technique. Eades *et al.* [10] suggest that random edge sampling can be useful because of its speed and simplicity. This gives support to the idea that random edge sampling could also be effective for speeding up spring force calculations.

3 EDGE SAMPLING ALGORITHMS

Force-directed algorithms are iterative, meaning that they repeatedly compute forces based on the current vertex positions in order to derive new vertex positions. Typically, these algorithms use all the edges in the graph to compute spring forces between each pair of incident vertices using Hooke’s law or a similar formulation. In contrast, the spring force calculations can be sped up by using sampling. This paper proposes two types of edge sampling algorithms for use in force-directed graph layout algorithms: (1) sample a fixed percentage of edges at each iteration, or (2) use a geometric series to determine the number of edges to sample. These are built as a plugin for v4 and v5 of the D3 JavaScript visualization library [4].

3.1 Percentage Edge Sampling

The percentage edge sampling algorithm uses a fixed percentage of edges at each iteration to compute spring forces. Instead of choosing a sample of edges with uniform probability over each edge, the algorithm uses a sliding window over the array of edges to select adjacent edges in the array, thereby taking advantage of data locality. The window length is equal to the percentage of edges that will be

*e-mail: robert.gove@twosixlabs.com

sampled. For example, setting the percentage to 50% and running the algorithm on a graph with 40 edges means the window size will be 20 edges. During the first iteration the algorithm will sample the first 20 edges, during the second iteration it will sample the last 20 edges, during the third iteration it will again sample the first 20 edges, etc. Given enough iterations, all edges will be sampled the same number of times.

To compensate for calculating the spring force of each edge less often, the algorithm scales up the magnitude of the force. The scale is inversely proportional to the fraction of edges in the sliding window. For example, setting the percentage to 50% or 30% results in the spring force being scaled by $1/0.5$ or $1/0.3$, respectively.

3.2 Geometric Series Edge Sampling

The geometric series edge sampling algorithm uses a sliding window similar to the percentage edge sampling algorithm. However, this algorithm uses a geometric series to determine the window size.

Force-directed graph layout algorithms often have a “cooling” or “temperature” parameter, denoted α . This serves to slow velocity in later iterations of the algorithm and help prevent oscillations. Typically, the initial value for α , denoted α_0 , is 1. Then at the next iteration, α is updated to be $\alpha_0(1-d)$, where $d = 1 - 0.001^{1/300}$ is the decay rate (0.001 is the threshold where the algorithm stops when $\alpha < 0.001$, and 300 is the default number of iterations). Generally, at iteration i , we have $\alpha_i = \alpha_{i-1}(1-d) = \alpha_0(1-d)^i = (1-d)^i$, because $\alpha_0 = 1$. This is the geometric series $\sum_{i=0}^{300} (1-d)^i$, which converges because $|1-d| < 1$. Specifically, $\sum_{i=0}^{300} (1-d)^i < \sum_{i=0}^{\infty} (1-d)^i = 1/(1-(1-d)) = 1/d \approx 43.93$.

Using this result, we can set the *geometric edge sampling* window size to $\alpha|E|$ at each iteration. This would therefore reduce the number of spring force calculations from $300|E|$ to about $43.93|E|$. It would also compute more spring forces at the beginning when the cooling parameter is large and fewer spring forces later when the cooling parameter is small. Importantly, even if the number of iterations were arbitrarily large, the number of spring force calculations would still be about $43.93|E|$ because of the geometric series.

Instead of computing $\alpha|E|$ spring forces at each iteration, the algorithm could calculate $(1-\alpha)|E|$ spring forces at each iteration. This is called the *reverse geometric edge sampling* algorithm.

To compensate for calculating the spring force of each edge less often, the geometric and reverse geometric edge sampling algorithms scale the spring force by $1/\alpha$ and $1/(1-\alpha)$, respectively.

4 EXPERIMENTS

To evaluate the edge sampling algorithms, this paper presents a series of experiments that compares using no edge sampling to five edge sampling algorithms: percentage edge sampling with 50%, 30%, and 10% window sizes; geometric edge sampling; and reverse geometric edge sampling. Each edge algorithm is tested in two force-directed graph layout algorithms: one with a Barnes-Hut [3] electric force approximation algorithm and the other with a random vertex sampling [16] electric force approximation algorithm. With each combination, this results in 12 pairs of force calculation algorithms. The layout algorithms use D3’s default settings, except for the algorithms using random vertex sampling which use a velocity decay of 0.2 per Gove’s recommendation [16].

4.1 Layout Readability Metrics

This experiment uses the 72 graphs with 1,000 or fewer vertices from the random vertex sampling experiments [16]. These graphs comprise several different types of graphs, including artificially generated graphs, character interaction graphs, ego networks, geometric graphs, software dependency graphs, and subway station connection graphs. This is important because research suggests that different graph types are topologically distinct and that artificially generated graphs are not topologically similar to social networks [17]. These

Figure 1: Means and 95% confidence intervals for readability metric for each edge sampling algorithm. Layout algorithms use Barnes-Hut (top) and Random Vertex Sampling (bottom) for electric forces.

graphs range in size from 4 to 1,000 vertices and 6 to 30,772 edges. Some graphs have many components, and the densest graph has a density of 0.27. See Gove [16] for additional information on the data.

This evaluation computed 20 layouts for each graph using the 12 force algorithm combinations. The experimental environment was NodeJS version 9.4.0 on a 2015-model MacBook Pro with a 3.1 GHz Intel Core i7 processor and 16 GB of RAM. Vertices were initially positioned in a phyllotaxis arrangement, and each vertex’s position in the arrangement was randomly assigned.

This evaluation uses the four readability metrics available in Greadability.js [14]: number of edge crossings, edge crossing angle, angular resolution (min), and angular resolution (dev). (See Dunne *et al.* [7] and Gove [15] for the definitions of these metrics.) This evaluation does not include other metrics, such as stress or standard deviation of edge length, due to concerns raised in other papers [9, 12, 13, 16, 24]. Some work has indicated that it is better not to optimize for a single graph readability metric [22]. For this reason, this evaluation averages the four readability metrics for each graph’s 20 layouts, and then computes the median of the 20 average values for each graph. This is used as the readability metric score for each layout algorithm on each graph and helps us understand when a layout algorithm performs well across many readability metrics.

This evaluation uses a within-subjects design, comparing the median readability metrics of each graph between the layout algorithms. When using Barnes-Hut to approximate electric forces, there is a significant difference in mean readability metric between the six edge algorithms (ANOVA, $F(5,355) = 26.17$, $p < 0.001$). Fig. 1 (top) shows the differences: Reverse geometric edge sampling and 10% edge sampling both are worse than no edge sampling, but geometric, 50%, and 30% are all comparable to no edge sampling.

We also see a significant difference in mean readability metric between the six edge algorithms when using random vertex sampling to approximate electric forces (ANOVA, $F(5,355) = 47.43$, $p < 0.001$). Fig. 1 (bottom) shows the differences: geometric and 10% edge sampling are substantially worse than no edge sampling. Although 50%, 30%, and reverse geometric edge sampling do not show significant differences from no edge sampling, 50% edge sampling has a nearly identical mean readability metric as no edge sampling on this data set.

The supplementary material contains an additional figure analyzing the edge sampling algorithms on each individual readability metric. For most readability metrics we see the same ranking of edge sampling algorithms as in Fig. 1.

Figure 2: Example graph layouts for the six edge algorithms using either Barnes-Hut (top) or random vertex sampling (bottom).

Figure 3: TVCG collaboration graph. Layouts using 50% or no edge sampling with either Barnes-Hut or random vertex sampling.

Figure 4: Runtime on large graphs ($|E| = 2|V|$).

4.2 Example Layouts

To better understand the graph layout quality, this section presents example graph layouts generated by each edge sampling algorithm paired with each electric force approximation algorithm (Barnes-Hut and random vertex sampling) on several graphs from the random vertex sampling experiments [16]. Fig. 2 shows layouts for the 4x4 grid, Les Miserables character interaction graph, Berlin subway connections graph, and Netscience collaboration graph. These layouts largely confirm the readability metric experiment. Reverse geometric edge sampling and 10% edge sampling both produce undesirable layouts in most cases. Geometric edge sampling and 30% edge sampling both produce adequate results when using Barnes-Hut, but the layouts can be unreadable when using random vertex sampling. 50% edge sampling appears quite similar to no edge sampling for both Barnes-Hut and random vertex sampling.

In large graphs with many components, such as the TVCG collaboration graph in Fig. 3, using random vertex sampling with 50% edge sampling appears to be less effective at separating components than using random vertex sampling with no edge sampling. For example, the graph contains 330 components, but we can only count 304 components in the layout using 50% edge sampling with random vertex sampling—the rest overlap the large component in the middle. We see a similar effect when using 50% vs. no edge sampling with Barnes-Hut, but the effect appears less noticeable.

4.3 Runtime

To evaluate the performance of each edge sampling algorithm as $|E|$ increases, Fig. 4 shows the runtime of each edge sampling algorithm on three sparse graphs with 20,000, 200,000 and 2,000,000 vertices. (The layout algorithms used random vertex sampling to approximate electric forces because it is faster than Barnes-Hut, which makes it easier to see small overall runtime differences.) The graphs were generated with $|E| = 2|V|$ random edges (without replacement). Keeping the proportion of edges fixed allows us to more easily understand the change in runtime performance as graphs get larger. The runtime improvement is quite small for graphs with 200,000 edges or fewer. On the graph with 2,000,000 edges, using the 50% edge sampling algorithm runs about 231 seconds faster (about 10% faster) than not using edge sampling.

As a baseline comparison, a layout algorithm using only random vertex sampling and no electric forces (i.e. equivalent to 0% edge sampling) took 2012 seconds on the graph with 2,000,000 edges, about 18 seconds faster than 10% edge sampling and about 394 seconds faster than no edge sampling.

5 DISCUSSION

The example layouts in Sect. 4.2 corroborate some of the layout readability metric experimental results in Sect. 4.1. For example, 10% random edge sampling had a significantly worse mean readability metric than no edge sampling, and the layouts in Fig. 2 mirror this finding. However, 30% edge sampling did not have a significantly worse mean readability metric than no edge sampling, but the layouts for Les Miserables and Netscience are both substantially worse when using random vertex sampling. This may indicate that even though the difference is not statistically significant, the effect size (0.03 sample mean difference) is enough to be meaningful in

practice. Or this could indicate that the readability metrics are not suitable for measuring the layout deficiencies shown here.

Overall, the runtime benefit of edge sampling is quite small compared to the runtime benefit of using electric force approximation algorithms like Barnes-Hut [28] or random vertex sampling [16]. Electric force approximation algorithms substantially improve runtime without impacting layout quality, but this does not appear to be the case with edge sampling. Even the 50% edge sampling algorithm had minor layout programs with the TVCG graph. Furthermore, the runtime on the largest graph analyzed in Sect. 4.3 indicates that calculating edge forces is only about 16% of the total runtime to calculate a force-directed layout; although it would be more for a denser graph. Considering these results, it appears that electric force approximations are more effective than spring force approximations at reducing runtime without compromising layout quality. The sharp degradation in layout quality may also indicate that spring forces are more important for producing good graph layouts than electric forces. Future work should address these questions more directly.

This paper did not examine the convergence rate of the edge sampling algorithms. Considering the similarity of 50% edge sampling to no edge sampling with respect to layout results and runtime, it seems unlikely that their convergence rates would be meaningfully different. The runtime benefit of edge sampling is only apparent on very large graphs, but the readability and quality metrics are quite slow ($O(|V|^2)$ time or worse) making it computationally difficult to effectively examine convergence rate. Future work should develop faster ways to measure layout quality to enable convergence rate analysis large graphs.

The edge sampling algorithms in this paper use a simple uniform sampling scheme. It could be possible that using a method to weight the sampling probabilities could result in better layout quality. However, calculating edge forces runs in linear time, whereas many weighting algorithms are super linear time, which could reduce the overall runtime instead of speed it up. Furthermore, some research suggests that weighting edges may not provide much improvement in quality [10]. However, future work could explore linear-time weighting algorithms to see if they provide some advantage for improving layout quality in this context.

6 CONCLUSION

This paper presented a family of edge sampling algorithms to reduce the runtime of spring force calculations in force-directed graph layout algorithms. Experiments show that spring force calculation is a small percent of the total graph layout runtime, and that edge sampling algorithm runtime improvements are modest compared to gains by using electric force approximation algorithms like Barnes-Hut, and the sampling algorithms tend to have a small to large reduction in layout quality. Although edge sampling algorithms may have limited value in practice, this research indicates that accurate spring forces may be more important for force-directed graph layout algorithms than accurate electric forces. This opens up an interesting research question that could have important implications for graph layout algorithm design.

REFERENCES

- [1] D. Archambault, T. Munzner, and D. Auber. TopoLayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, 2007.
- [2] A. Arleo, W. Didimo, G. Liotta, and F. Montecchiani. A Million Edge Drawing for a Fistful of Dollars. In *International Symposium on Graph Drawing and Network Visualization*, pp. 44–51, 2015.
- [3] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force calculation algorithm. *Nature*, 324:446–449, 1986.
- [4] M. Bostock, V. Ogievetsky, and J. Heer. Data-Driven Documents. *IEEE TVCG*, 17(12):2301–2309, 2011.
- [5] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67–90, 1995.
- [6] M. Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *Proceedings of IEEE Visualization*, pp. 127–132, 1996.
- [7] C. Dunne, S. I. Ross, B. Shneiderman, and M. Martino. Readability metric feedback for aiding node-link visualization designers. *IBM Journal of Research and Development*, 59(2/3):14:1–14:16, 2015.
- [8] T. Dwyer, B. Lee, D. Fisher, K. I. Quinn, P. Isenberg, G. Robertson, and C. North. A comparison of user-generated and automatic graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):961–968, 2009.
- [9] P. Eades, S.-H. Hong, A. Nguyen, and K. Klein. Shape-Based Quality Metrics for Large Graph Visualization. *Journal of Graph Algorithms and Applications*, 21(1):29–53, 2017.
- [10] P. Eades, Q. Nguyen, and S.-H. Hong. Drawing Big Graphs using Spectral Sparsification. In *International Symposium on Graph Drawing and Network Visualization*, pp. 272–286, 2017.
- [11] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.
- [12] E. R. Gansner, Y. Koren, and S. C. North. Graph drawing by stress majorization. In *International Symposium on Graph Drawing*, pp. 239–250, 2004.
- [13] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203–1233, 1999.
- [14] R. Gove. Greadability.js, 2018.
- [15] R. Gove. It pays to be lazy: Reusing force approximations to compute better graph layouts faster. In *11th Forum Media Technology*, pp. 43–51, 2018.
- [16] R. Gove. A Random Sampling $O(n)$ Force-calculation Algorithm for Graph Layouts. *Computer Graphics Forum*, 38(3), 2019.
- [17] R. Gove. Gragnostics: Fast, interpretable features for comparing graphs. In *Conference on Information Visualization*, pp. 201–209, 2019.
- [18] L. F. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. PhD thesis, Yale University, New Haven, CT, USA, 1987.
- [19] S. Hachul and M. Jünger. Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm. In Pach J., ed., *Lecture Notes in Computer Science*, chap. Graph Drawing 2004, pp. 285–295. Springer-Verlag, Berlin, 2005.
- [20] Y. Hu. Efficient, High-Quality Force-Directed Graph Drawing. *Mathematica Journal*, 10(1):37–71, 2005.
- [21] W. Huang. Using eye tracking to investigate graph layout effects. In *Asia-Pacific Symposium on Visualisation*, pp. 97–100, 2007.
- [22] W. Huang, P. Eades, S. H. Hong, and C. C. Lin. Improving force-directed graph drawings by making compromises between aesthetics. In *VL/HCC*, pp. 176–183, 2010.
- [23] S. G. Kobourov. Force-Directed Drawing Algorithms. In R. Tamassia, ed., *Handbook of Graph Drawing and Visualization*, chap. 12, pp. 383–408. Chapman and Hall/CRC, Boca Raton, 1 ed., 2016.
- [24] J. F. Kruiger, P. E. Rauber, R. M. Martins, A. Kerren, S. Kobourov, and A. C. Telea. Graph Layouts by t-SNE. *Computer Graphics Forum*, 36(3):283–294, 2017.
- [25] F. Lipp, A. Wolff, and J. Zink. Faster force-directed graph drawing with the well-separated pair decomposition. In *International Symposium on Graph Drawing and Network Visualization*, pp. 52–59, 2015.
- [26] A. Morrison, G. Ross, and M. Chalmers. Fast multidimensional scaling through sampling, springs and interpolation. *Information Visualization*, 2(1):68–77, 2003.
- [27] M. Pohl, M. Schmitt, and S. Diehl. Comparing the Readability of Graph Layouts using Eyetracking and Task-oriented Analysis. In *Computational Aesthetics in Graphics, Visualization, and Imaging*, pp. 49–56, 2009.
- [28] A. Quigley and P. Eades. FADE: Graph drawing, clustering, and visual abstraction. In *International Symposium on Graph Drawing*, pp. 197–210, 2000.
- [29] F. Van Ham and B. E. Rogowitz. Perceptual organization in user-generated graph layouts. *IEEE TVCG*, 14(6):1333–1339, 2008.