

Planar Preprocessing for Spring Embedders

J. Joseph Fowler and Stephen G. Kobourov

Department of Computer Science
University of Arizona

Technical Report 12-03
September 4, 2012

Abstract. Spring embedders are conceptually simple and produce straight-line drawings with an undeniable aesthetic appeal, which explains their prevalence when it comes to automated graph drawing. However, when drawing planar graphs, spring embedders often produce non-plane drawings, as edge crossings do not factor into the objective function being minimized. On the other hand, there are fairly straight-forward algorithms for creating plane straight-line drawings for planar graphs, but the resulting layouts generally are not aesthetically pleasing, as vertices are often grouped in small regions and edges lengths can vary dramatically. It is known that the initial layout influences the output of a spring embedder, and yet a random layout is nearly always the default. We study the effect of using various plane initial drawings as an inputs to a spring embedder, measuring the percent improvement in reducing crossings and in increasing node separation, edge length uniformity, and angular resolution.

1 Introduction

Some of the most flexible algorithms for drawing simple undirected graphs belong to a class known as *force-directed algorithms*. Also called *spring embedders*, such algorithms compute the layout of a graph using only information contained within the structure of the graph itself, rather than relying on domain-specific knowledge. Graphs drawn with such algorithms tend to be symmetric and have uniform edge lengths with good vertex distribution. As these are often desirable goals for the readability of a graph [18], it is no surprise that force-directed algorithms are the most commonly available tool for drawing graphs, networks, and diagrams. From family trees drawn by amateur genealogists to chemical molecules and Facebook friendship networks, the overwhelming majority of graphs that are drawn with such tools are small and sparse.

In general, force-directed methods define an objective function which maps each graph layout into a number in \mathbb{R}^+ representing the energy of the layout. This function is defined so that low energies correspond to layouts in which adjacent nodes are near some pre-specified distance from each other, and in which non-adjacent nodes are well-spaced. Typically, force-directed algorithms are initialized with a random layout and iteratively move vertices so as to find a (often local) minimum of the objective function. Therefore, different initial layouts can influence the quality of the final layouts.

If the input graph happens to be planar, force-directed methods may reduce the edge crossings present in the initial random layout, although the objective function does not explicitly consider them. At the same time it is easy to compute an initial *plane drawing* in linear time and on an integer grid [16, 20]. Even for planar graphs, the running time of most force-directed algorithms is quadratic given that all vertex pairs are considered. Hence, adding an efficient preprocessing step that computes a plane (instead of random) initial layout, would not significantly increase the overall running time of a standard force-directed tool. We study the effect of such planar preprocessing on the number of crossings, node separation, edge lengths, and angular resolution.

1.1 Related Work

An early force-directed method by Eades [10] models edges as springs obeying Hooke’s Law. A popular variant is that of Fruchterman and Reingold (FR) [12], who model the problem in terms of a strong nuclear force attracting two protons within the atomic nucleus at close range, but with an electrical force repelling them at a further range. Alternatively, forces between the nodes can be computed based on their graph theoretic distances, determined by the lengths of shortest paths between them. The algorithm of Kamada and Kawai (KK) [15] uses spring forces proportional to graph theoretic distances, and is in effect a variant of multi-dimensional scaling (MDS).

Modifications to the basic force-directed functionality, with the aim of improving the layout quality for planar graphs, have also been considered. Harel and Sardas [14] use an initial plane embedding and then apply simulated annealing while not introducing any crossings. They use an $O(n^2)$ -time heuristic to find a large face as the outerface and an $O(n^3)$ -time step to “center” the vertices in the drawing. Overall this method significantly improves the aesthetic quality of the planar layout, but at the expense of a significant increase in running time.

Bertault [1] describes a force-directed algorithm, PrED, that avoids introducing edge crossings by using repulsive forces between the n vertices and the m edges in the graph. This approach also leads to an increase in running time of $O(n^2 + nm)$ for each iteration. Moreover, since adjustments are much more restricted in this model, many more iterations can be required to reach a low energy state. Recently, Simonetto *et al.* improved this approach with the faster algorithm ImPrEd [21]. Tunkelang [22] studies a variant of the Fruchterman-Reingold method using quad trees to approximate forces at a distance in order to compute a gradient, which took $O(m + n \log n)$ time. Even though vertex-edge repulsive forces are used, it is not clear whether this results in overall reduction of edge crossings.

Brandes and Pich [3] show that layouts obtained with MDS are better at preserving relative distances than those obtained by force-directed algorithms. While they do not consider the number of edge crossings explicitly, one could argue that an algorithm that better preserves distances in the graph could lead to fewer edge crossings overall.

Using a greedy heuristic of picking edges from low to high betweenness (the likelihood of an edge participating in some shortest path), van Ham and Wattenberg [23] extracted a spanning planar subgraph with a low overall betweenness consisting of 80% of the edges in their real-world graph having a small diameter. By laying out this planar subgraph with a straight-line force-directed algorithm and drawing the remaining edges with curved arcs, they construct a drawing where clusters are clearly separated.

Other applications of force-directed methods to polylines drawings include Didimo, Liotta, and Romeo [7] and Dwyer *et al.* [8, 9]. Didimo *et al.* developed a customized force-directed algorithm to augment planarization-based approaches, e.g., orthogonal and poly-line layouts with bends in order to achieve drawings with fewer crossings than standard straight-line force-directed algorithms for a sampling of random graphs over a range of edge densities and for the Rome graph library. Their force-directed algorithm works by restricting the range of each force acting on vertex so that no additional crossings are introduced, which can alter the topology since crossings can potentially be eliminated from the original layout. Alternatively, Dwyer *et al.* showed how to use *P-stress*, a bend-point invariant goal function, to construct topology preserving constrained graph layouts, where edges form polylines that can bend like rubber bands.

2 Hypothesis and Methodology

There are many specialized drawing algorithms that guarantee crossings-free layouts for certain classes of graphs: binary trees, arbitrary trees, outerplanar graphs, general planar graphs. However, most popular graph visualization tools (e.g., GraphViz, yEd, Gephi) do not contain implementations of such algorithms but rather use a spring embedder and do not test whether the input graph is planar. At the same time, many real-world examples (e.g., molecules in chemistry, small business hierarchies, genealogical relationships) are indeed planar and spring embedders are known for failing to find crossings-free layouts.

Our main goal is to gauge to what degree, for a given planar graph, that a plane embedding preprocessing step can reduce the number of edge crossings in the final drawing. Similarly, we would like to assess the effect, if any, the preprocessing step has on node separation, edge lengths and angular resolution.

It is generally believed that the quality of the final layout of a spring embedder can be improved by starting with a “better” layout than a random one. However, to the best of our knowledge, there are no results studying the effect that such a preprocessing step can have on reducing the number of crossings while maintaining other aesthetic qualities. With this in mind, we took three standard planar layout algorithms and four off-the-shelf spring embedders and tried to study these effects.

2.1 Data-sets

We use six libraries of connected planar graphs for our experiments:

- I **Prüfer trees** (PRUFER) a set of uniformly sampled labeled trees obtained by using Prüfer’s algorithm [17];
- II **Random trees** (R-TREES) a set of random trees with unbounded maximum degree constructed by adding new pendant edges to randomly selected nodes in the tree constructed so far;
- III **Fusy graphs** (FUSY) a set of uniformly sampled unlabeled graphs created using the Fusy generator [13];
- IV **Expansion graphs** (EXPAN) a set of random triconnected planar graphs using the expansion method that performs $n - 4$ split operations (starting with a K_4) on randomly selected nodes that randomly distributes neighbors between split nodes;

| Library | Size | n | | | | | m | | | | | Average Degree Percentile | | | | | | |
|---------|------|-----|-----|------|-----|------|-----|-----|-------|-----|------|---------------------------|------|------|------|------|------|------|
| | | min | max | avg | med | std | min | max | avg | med | std | 0% | 5% | 25% | 50% | 75% | 95% | 100% |
| PRUFER | 910 | 10 | 100 | 55.0 | 55 | 26.3 | 9 | 99 | 54.0 | 54 | 26.3 | 1.80 | 1.86 | 1.94 | 1.96 | 1.97 | 1.98 | 1.98 |
| R-TREES | 910 | 10 | 100 | 55.0 | 55 | 26.3 | 9 | 99 | 54.0 | 54 | 26.3 | 1.80 | 1.86 | 1.94 | 1.96 | 1.97 | 1.98 | 1.98 |
| FUSY | 910 | 10 | 100 | 55.0 | 55 | 26.3 | 11 | 230 | 116.7 | 116 | 58.7 | 2.20 | 3.60 | 4.00 | 4.21 | 4.35 | 4.53 | 4.79 |
| EXPAN | 910 | 10 | 100 | 55.0 | 55 | 26.3 | 18 | 240 | 126.0 | 126 | 61.9 | 3.60 | 4.19 | 4.43 | 4.55 | 4.67 | 4.81 | 5.18 |
| ROME | 3279 | 10 | 89 | 25.8 | 24 | 11.8 | 9 | 103 | 30.2 | 28 | 14.2 | 1.80 | 2.00 | 2.17 | 2.32 | 2.46 | 2.77 | 3.85 |
| AT&T | 854 | 10 | 100 | 29.0 | 23 | 19.4 | 9 | 120 | 32.7 | 26 | 21.3 | 1.80 | 1.85 | 2.00 | 2.22 | 2.47 | 3.00 | 4.74 |

Table 1: Data-set statistics showing the minimum (min), maximum (max), average (avg), median (med), and standard deviation (std) of the number of vertices n and the number of edges m of each library with the percentile distribution for the average degree.

- V **Rome graphs (ROME)** a subset of the undirected Rome graphs from the GDFToolkit consisting of all connected planar graphs; and
- VI **AT&T graphs (AT&T)** a subset of the directed AT&T graphs (also known as the Graph Catalog) consisting of all connected planar graphs.

Libraries I–IV were constructed to each have 10 graphs per value n , whereas, V and VI have a variable number of graphs per value of n ; see Table 1. Both tree libraries I and II have the lowest median average degree of 1.96, while the other two constructed libraries III and IV have higher median average degrees of 4.21 and 4.55, respectively, whereas the pre-existing libraries V and VI also have low median average degrees of 2.32 and 2.22, respectively. Hence, libraries I, II, V, and VI all have relatively low edge-density, while III and IV both have relatively high edge-density.

2.2 Tools

While there are many popular graph drawing tools (e.g., GraphViz, yEd, Gephi), most of them do not contain algorithms for planar graphs. With this in mind, we use the Open Graph Drawing Framework (OGDF) [5], together with additional code and scripts to generate libraries I–IV, to compute the graph aesthetics metrics, and to perform the experiments. Our code is written in C++ and is available, along with all the graph libraries and experimental data, at <http://planarpreprocessing.cs.arizona.edu>. All figures with plots were drawn using `gnuplot` using the ‘`acspline`’ smoothing function.

2.3 Initial Placers

In addition to our hand-coded purely random layout (`RANDOM`), OGDF includes the following three straight-line drawing algorithms for planar graphs:

- (i) `FPPLayout (FPP)` implements the algorithm of de Fraysseix, Pach and Pollack [6] for plane drawing on a grid of size $(2n - 4) \times (n - 2)$, obtained by augmenting the graph until it is fully triangulated and then computing a canonical ordering and incrementally placing the vertices in this order;
- (ii) `SchnyderLayout (SCHNYDER)` implements the algorithm of Schnyder [20] for plane drawing on an integer grid of size $(n - 2) \times (n - 2)$, obtained by augmenting the graph until it is fully triangulated, partitioning the interior edges into three trees and computing barycentric coordinates for the vertices;
- (iii) `PlanarStraightLayout (KANT)` implements an improved version of the algorithm of Kant [16] for plane drawing on a grid of size at most $(2n - 4) \times (n - 2)$, obtained by augmenting the graph until it is triconnected and then using an appropriate modification of the FPP algorithm to compute coordinates for the vertices.

Using `RANDOM` as an initial placer is equivalent to have no preprocessing done (given that is what most spring embedders use by default), while using any of the other placers constitutes adding a preprocessing step.

2.4 Embedders

OGDF also includes the following four distinct energy-based spring embedding algorithms for general graphs:

- (1) `SpringEmbedderFR` (**FR**) implements the force-directed layout algorithm by Fruchterman-Reingold [12] using a grid-variant of the algorithm to speed up the computation of repulsive forces (using the defaults: (i) 400 iterations, (ii) has small initial random perturbations, (iii) minimum distance of 20 between connected components, and (iv) automatic scaling),
- (2) `SpringEmbedderFRExact` (**FRE**) provides an alternative implementation of the **FR** spring embedder with exact force computations (using the defaults: (i) 1000 iterations, (ii) has small initial random perturbations, (iii) minimum distance of 20 between connected components, (iv) cooling factor of .9, (v) ideal edge length of 10, and (v) tolerance factor of 0.01, which is the fraction of ideal length below which convergence is achieved);
- (3) `SpringEmbedderKK` (**KK**) implements that **MDS** spring embedder of Kamada and Kawai [15] (using the defaults: (i) stop tolerance of 0.0001 for energy level, (ii) maximum number of iterations is based on input graph, (iii) all edges have equal length, and (iv) desirable edge length is proportional to the area of the display grid divided by the maximum distance between two nodes in initial layout); and
- (4) `StressMajorization` (**SM**) implements simple stress majorization by Pich [4] that allows radial constraints based on shortest path distances (using the defaults: (i) 50 iterations and (ii) automatic scaling).

Aesthetics: In addition to counting crossings c of G , we consider the following three aesthetic parameters for a drawing of a graph on n nodes and m edges, which have been shown to have an effect upon human understanding of a graph; see e.g., Purchase [18]:

- (a) *Minimum angle aesthetic metric* ma from [19], is based on d_{ma} , the average deviation of adjacent incident edge angles from the ideal minimum angle:

$$d_{ma} = \frac{1}{n} \sum_{i=1}^n \left| \frac{\vartheta_i - \theta_{i \min}}{\vartheta_i} \right|$$

where ϑ_i is the *ideal minimal angle* at the i th node, namely $\vartheta_i = \frac{360^\circ}{\text{degree}(v_i)}$ and $\theta_{i \min}$ is the actual minimum angle between the incident edges at the i th node. Then ma is defined as $ma = 1 - d_{ma}$ so that $0 \leq ma \leq 1$, where $ma = 0$ indicates that every node in the drawing has a zero minimum angle (*i.e.*, two overlapping incident edges) and $ma = 1$ indicates “perfect angular resolution” in which case every angle is optimal as in Lombardi drawings.

- (b) *Edge lengths aesthetic metric* el is based on d_{el} , the average percent deviation of edge lengths using a mean central tendency:

$$d_{el} = \frac{1}{m} \sum_{j=1}^m \left| \frac{|e_j| - |e|_{avg}}{\max\{|e|_{avg}, |e|_{max} - |e|_{avg}\}} \right|$$

where $|e_j|$ is the length of the j th edge, $|e|_{avg}$ is the average edge length, and $|e|_{max}$ is the maximum edge length. Dividing by the $\max\{|e|_{avg}, |e|_{max} - |e|_{avg}\}$ guarantees that $0 \leq d_{el} \leq 1$, so that $el = 1 - d_{el}$ will be bounded $0 \leq el \leq 1$, where $el = 0$ could indicate that half the edges were all short while other half of the edges were all long while $el = 1$ indicates all the edges have the same length.

(c) *Node separation aesthetic metric* ns is based on d_{sr} , the average percent deviation of separating rectangles¹ using a mean central tendency:

$$d_{sr} = \frac{1}{n} \sum_{i=1}^n \left| \frac{|v_i| - |v|_{avg}}{\max\{|v|_{avg}, |v|_{max} - |v|_{avg}\}} \right|$$

¹ These are the n axis-aligned, smallest-perimeter rectangles contained within the bounding box of all nodes, such that each side of the rectangle is either incident to the neighbor(s) of its node or coincides with a side of the bounding box.

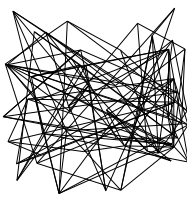
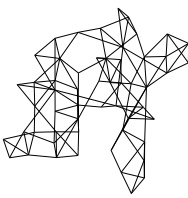
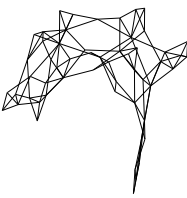
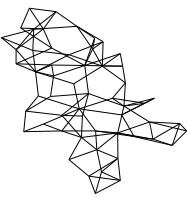

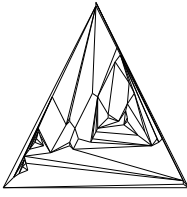
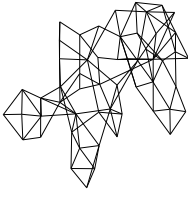
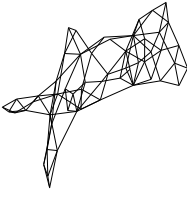
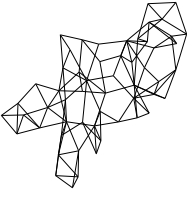
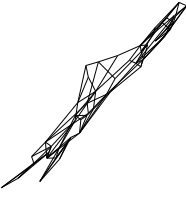
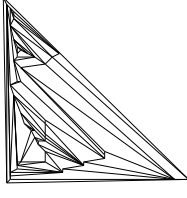
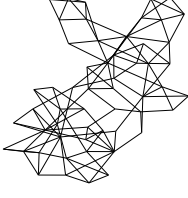

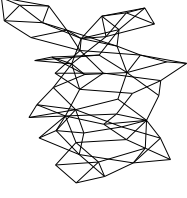
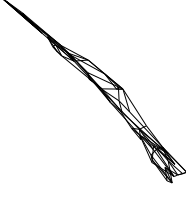
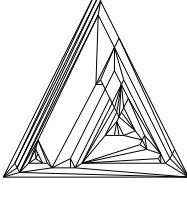
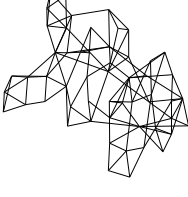
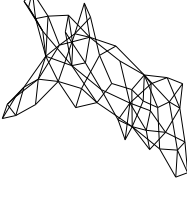
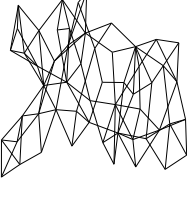
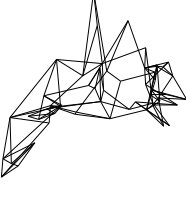
| Initial Placer | Spring Embedder | | | | |
|----------------|---|---|---|--|--|
| | None | FR | FRE | KK | SM |
| RANDOM |  c=1573, ma=0.26, el=0.59, ns=0.77 |  c=94, ma=0.40, el=0.74, ns=0.82 |  c=66, ma=0.41, el=0.72, ns=0.81 |  c=68, ma=0.40, el=0.67, ns=0.84 |  c=182, ma=0.26, el=0.65, ns=0.66 |
| FPP |  c=0, ma=0.34, el=0.44, ns=0.66 |  c=84, ma=0.42, el=0.73, ns=0.78 |  c=61, ma=0.51, el=0.69, ns=0.79 |  c=64, ma=0.46, el=0.74, ns=0.85 |  c=92, ma=0.26, el=0.59, ns=0.73 |
| SCHNYDER |  c=0, ma=0.38, el=0.45, ns=0.64 |  c=97, ma=0.49, el=0.71, ns=0.80 |  c=65, ma=0.49, el=0.70, ns=0.82 |  c=77, ma=0.40, el=0.73, ns=0.82 |  c=107, ma=0.22, el=0.65, ns=0.67 |
| KANT |  c=0, ma=0.42, el=0.42, ns=0.64 |  c=91, ma=0.46, el=0.71, ns=0.76 |  c=56, ma=0.49, el=0.69, ns=0.80 |  c=63, ma=0.47, el=0.70, ns=0.84 |  c=166, ma=0.31, el=0.65, ns=0.65 |

Table 2: Sample layouts of a representative 55-node graph with 121 edges from EXPAN library where each row uses a different placer and each column uses a different spring embedder (except for the first column using only the placer).

where $|v_i|$ is the *perimeter of the smallest rectangle separating the i th node from its nearest neighbors*; $|v|_{avg}$ is the average over all nodes; and $|v|_{max}$ is the maximum over all nodes. Given that $0 \leq d_{sr} \leq 1$, then $n\mathfrak{s} = 1 - d_{sr}$ will also be bounded $0 \leq n\mathfrak{s} \leq 1$. Then $n\mathfrak{s} = 0$, could indicate that half the nodes were relatively close to their nearest neighbors while the other half of the nodes were far from their nearest neighbors, whereas $n\mathfrak{s} = 1$ indicates the nodes are more uniformly distributed.

2.5 General methodology

For each graph used in our experiments, we evaluated the above four aesthetic metrics on layouts produced by each of the four initial placers and by each placer-embedder pairs, where the placer-generated layout is used as the initial layout for the embedder; see Table 2. (and Tables 3–7 in the appendix). To reduce the noise of our data, we ran each placer not once, but five times, to generate five distinct layouts of the same graph, which we then treated as a separate data point (*i.e.*, as though each of the five layouts were from five different graphs). Thus, for a given library such as PRUFER, with 10 graphs per value of n , this gave us 50 total data points per value of n . To obtain a value for a particular aesthetic metric, we took the median of the metric over all 50 layouts. When comparing the overall results of using a planar preprocessor, such as determining the percent reduction in the number of crossings in going from (RANDOM/FR) to (FPP/FR), we took the median of the percent reduction in crossings between each pair of layouts before and after the preprocessing step was included.

3 Experiments and Results

For each of the graph libraries, and for each spring embedder (FR, FRE, KK, SM) we ran experiments to determine the effect of planar preprocessing (using FPP, SCHNYDER, or KANT), compared against using RANDOM as an initial placer, on the aesthetic criteria (crossings, edge length, node separation, angular resolution). We first analyze the placers and embedders separately, before turning to combining the two.

3.1 Placer Aesthetics

Figure 1 provides a comparison between the four initial placers. Lower y -values for the left column of plots indicates fewer crossings, which is desirable. Higher y -values for the remaining three columns indicate more desirable aesthetic values for minimum angles, edge lengths, and node separation. As the only non-planar embedder, RANDOM has the highest number of crossings. It also has the worst angular resolution, but provides the best edge length and node separation. KANT has the best angular resolution and is consistently second in the other metrics, with 15-30% better angular resolution and about 20% more uniform edge lengths and node separation than the other planar placers. SCHNYDER and FPP have nearly identical aesthetics (within 5%) for all the graphs, with the first slightly better with angular resolution and node separation, and the second better with edge lengths. In summary, among the three planar placers, KANT has the best overall aesthetics.

3.2 Embedder Aesthetics

Figure 2 provides a comparison between the spring embedders, when using RANDOM as an initial placer. With no preprocessing, KK has the least number of crossings producing almost planar layouts for the tree and AT&T libraries. FR and FRE have similar performance, except

that the exact variant produces 25-50% fewer crossings across all six libraries. SM performs noticeably worse than the other three with up to 40% worse angular resolution and up to 25% worse edge lengths and node separation (although it is the fastest among the embedders). In summary, among the four embedders, KK has the best overall aesthetics when no preprocessing is done.

Brandenburg *et al.* [2] did an experimental study comparing straight-line graph drawing algorithms including variants of FRE and KK implemented in the GraphED framework. Their

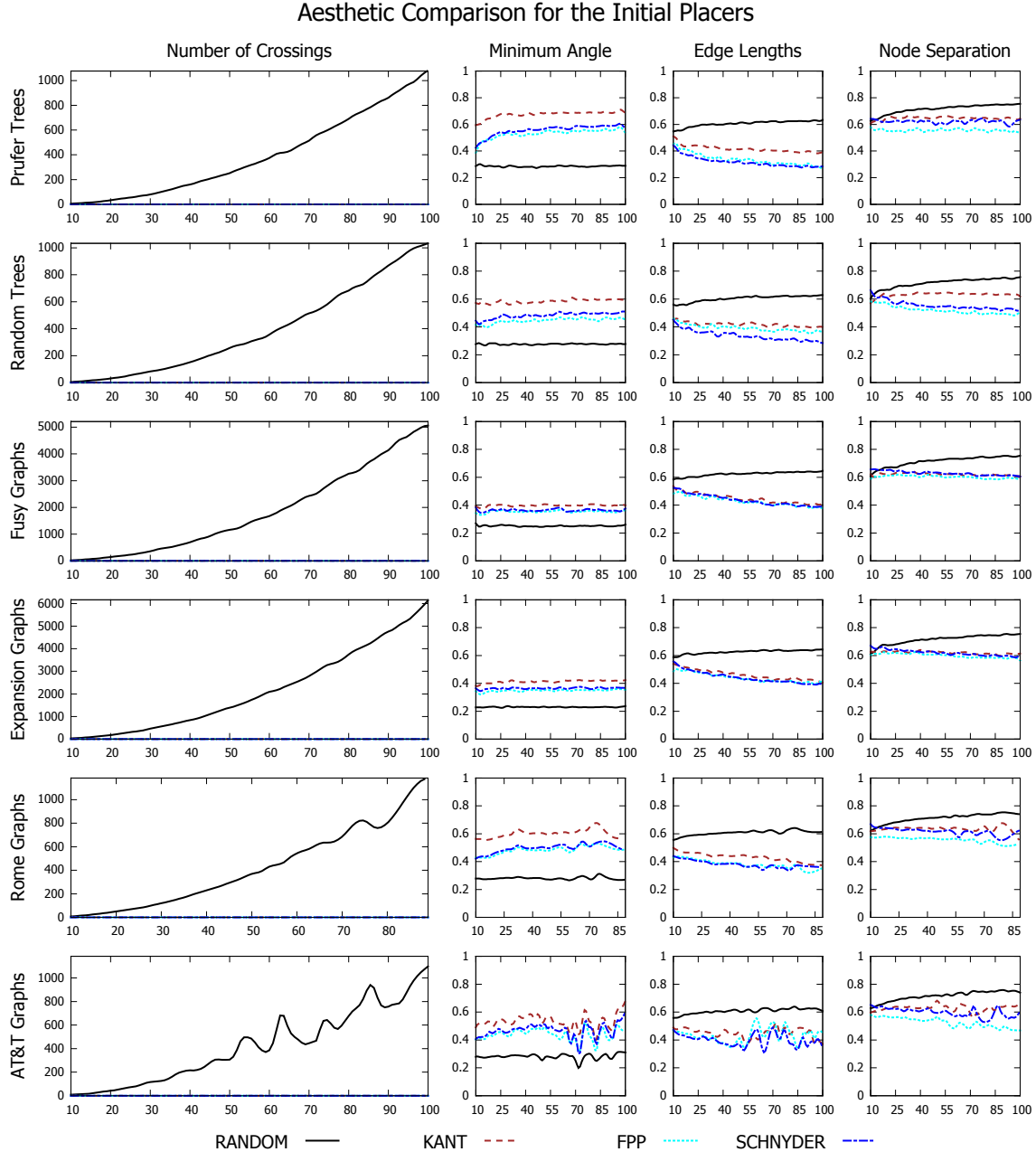


Fig. 1: Median of aesthetic metrics vs. number of nodes for each initial placer where values closer to 1 are better for the right three columns.

results confirm ours that KK is in general superior to FRE in terms of edge lengths and node distribution. While their variant of KK was shown to be more efficient, it should be noted that it only performs $10 * n$ iterations (regardless of the graph structure), rather than iterating until the energy function converges to a preset threshold as is the case in the standard KK algorithm used in OGDF. While $10 * n$ iterations may be sufficient for the smaller sparse graphs studied by Brandenburg *et al.*, this is not the case in general. For instance, for denser graphs in the FUSY and EXPAN libraries with a 100 edges, we found FRE to be up to 20 times faster than KK.

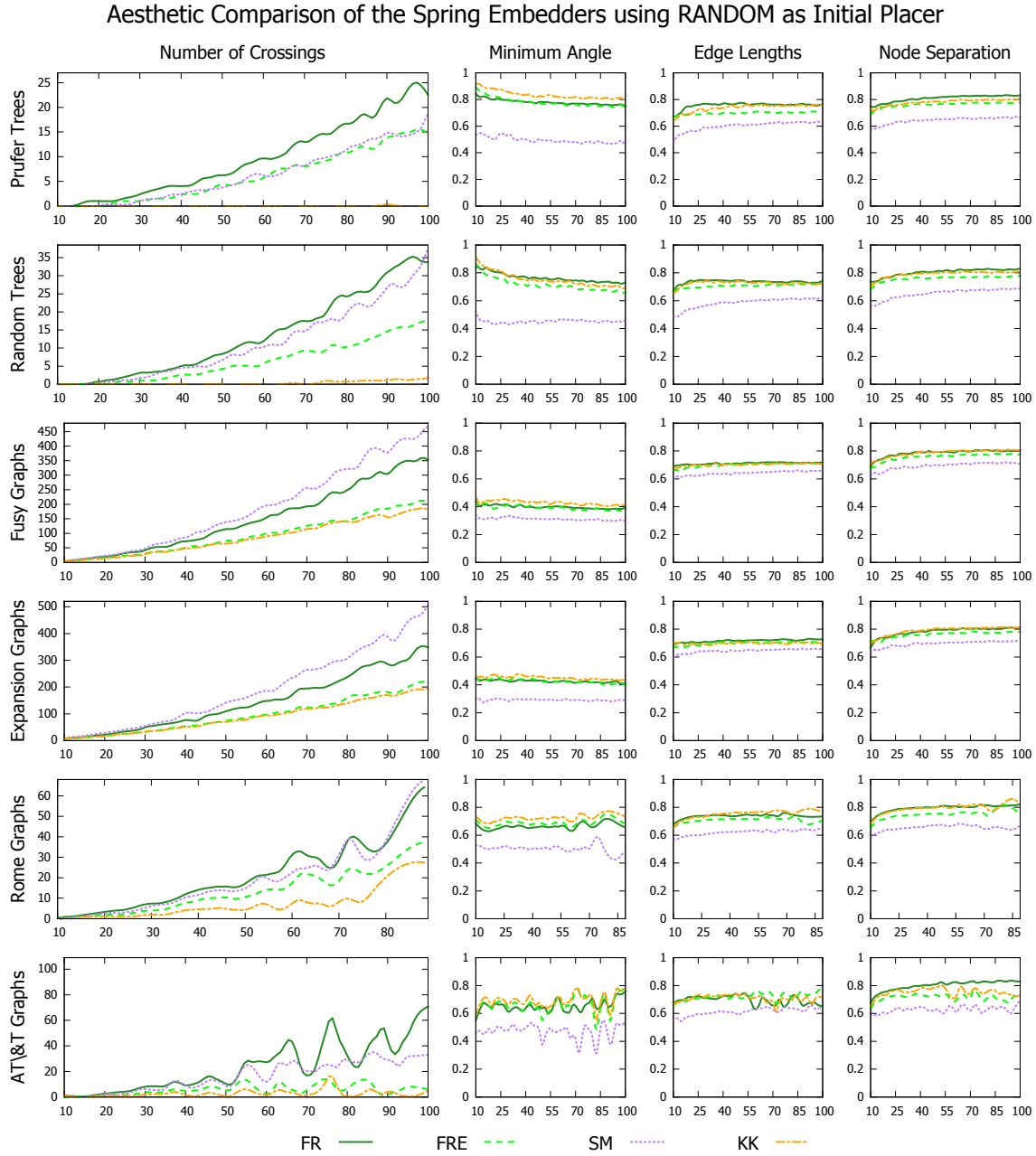


Fig. 2: Median of aesthetic metrics vs. number of nodes for each spring embedder using random layout where values closer to 1 are better for the right three columns.

3.3 Effect of Planar Preprocessing with KANT

Figure 3 shows the percent improvement in each aesthetic metric when KANT is used for planar preprocessing. The x -axis shows the number of vertices and the y -axis shows either the median number of crossings or the median of the normalized 0-to-1 value for the other metrics. High y -values in the left column indicate more crossing reduction, which is desirable. High y -values or near-zero y -values in the remaining columns indicate consistent improvement or no discernible effect with regard to that column's metric.

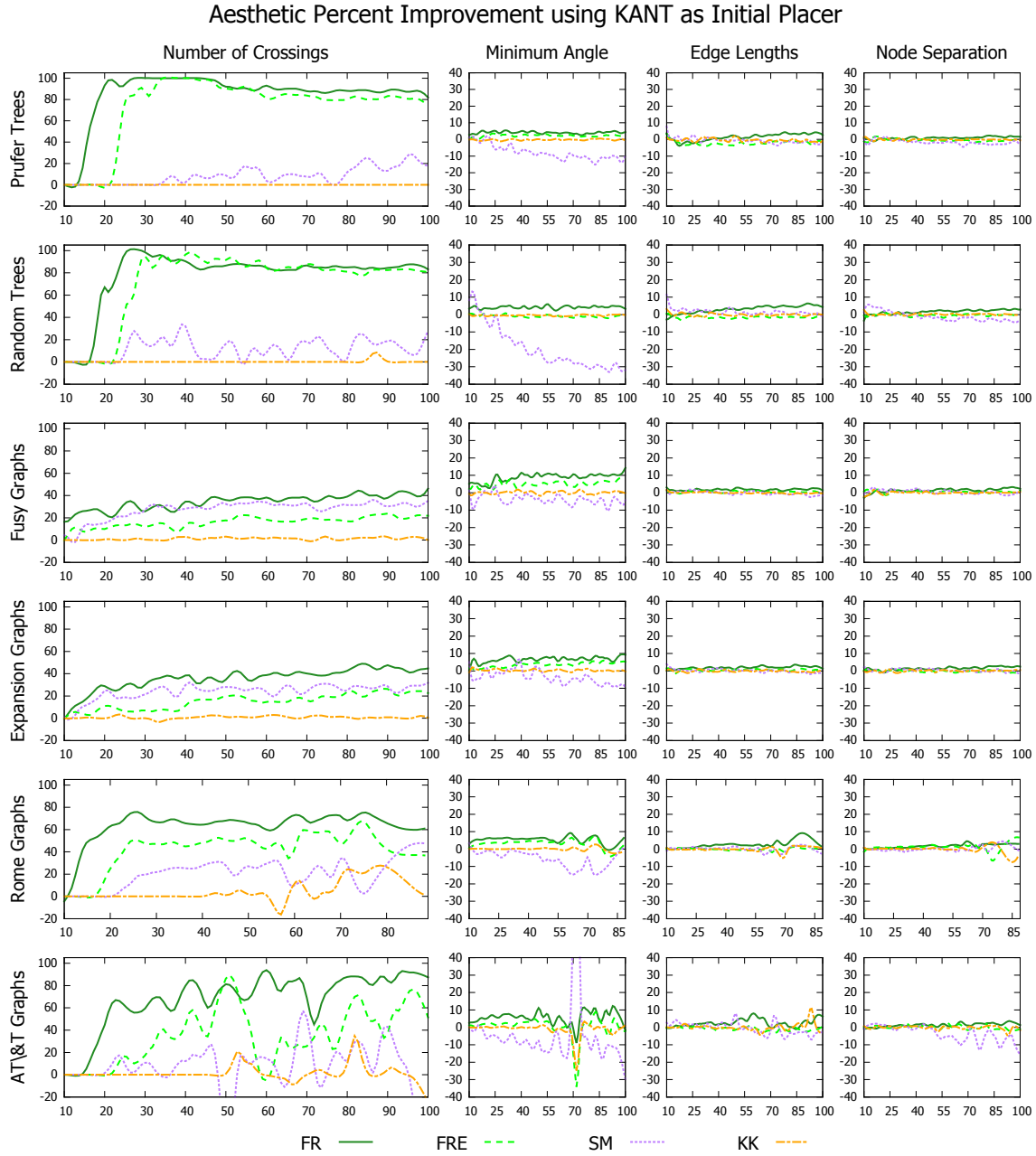


Fig. 3: Percent improvement in aesthetic metrics vs. number of nodes for spring embedders using KANT where higher, non-negative percentage is generally better.

Preprocessing with KANT clearly reduced crossings. Moreover, this improvement does not come at the cost of negatively impacting any of the other aesthetic metrics. From the last three columns in Fig. 3, we can see a remarkable lack of degradation in any of the three metrics for FR and FRE. The only exception is SM which shows worse angular resolution (which was already the worst even without preprocessing).² The effect of preprocessing is negligible for

² It is worth noting here that SM itself is more of a pre-processing step than a standard embedder.

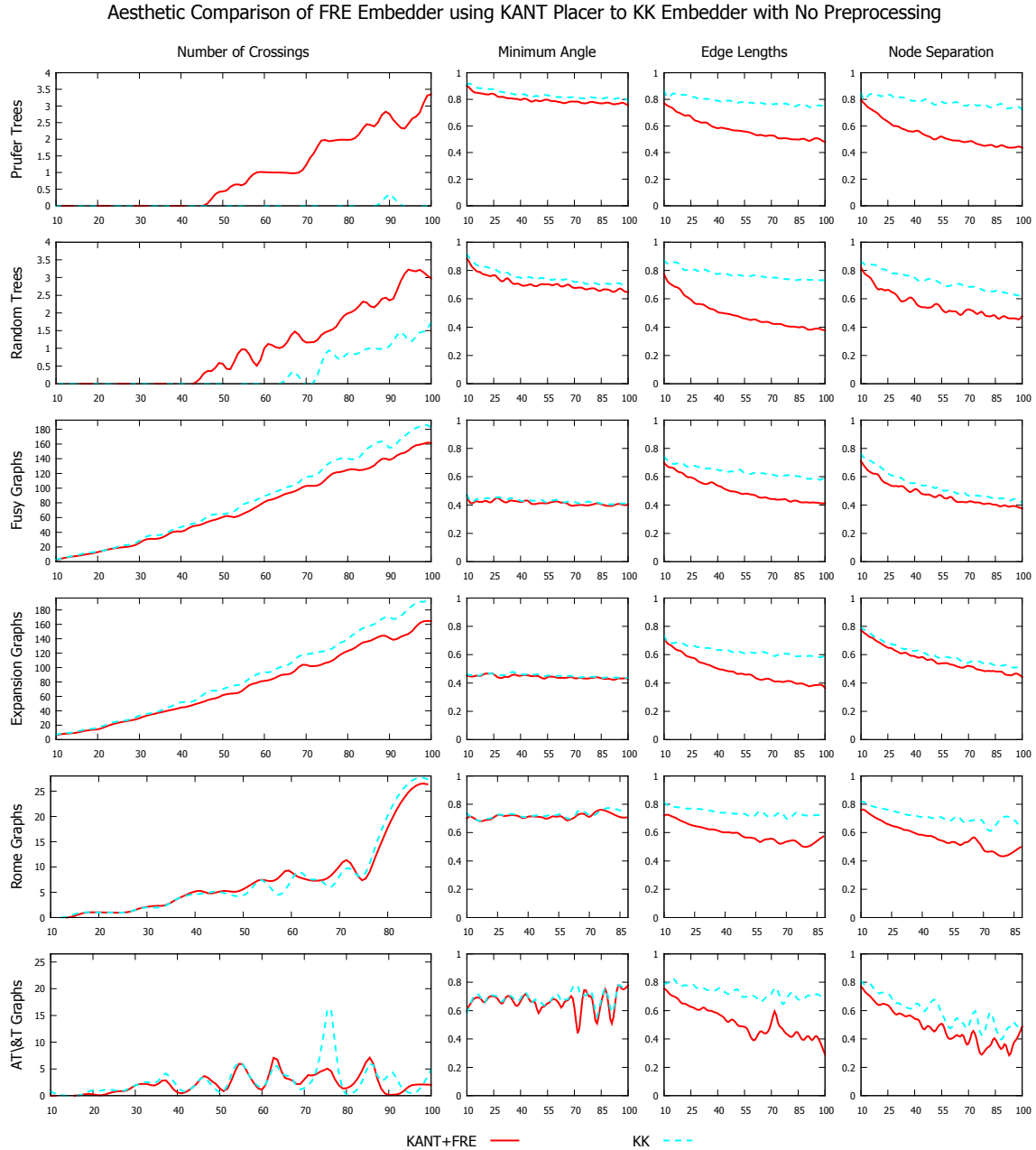


Fig. 4: Median of aesthetic metrics vs. number of nodes comparing KANT+FRE to KK (with no preprocessing) where values closer to 1 are better for the right three columns.

KK. In summary, using KANT as a preprocessing step for FR and FRE resulted in significant reduction in crossings, while not negatively impacting the other aesthetics.

Specifically, our results indicate that KANT+FRE is the best placer+embedder pair. In fact, the combination is good enough to compare directly with the best (but much more expensive) KK embedder. The two approaches yield very similar results for all our graph libraries. KK is slightly better with 5 to 10 percent fewer crossings. On the other hand, KANT+FR is slightly better with 5 to 10 percent better edge lengths and node separation. Both approaches provide nearly identical minimum angle metrics; see Figure 4 that directly compares KANT+FRE to KK.

We would like to point out that most spring embedders currently in use are based on implementations very similar to FR and FRE, as implementations of KK are significantly more computationally expensive. Our experiments indicate that popular spring embedders might obtain nearly as good results as KK for planar graphs, with the help of a linear-time planar preprocessing step such as KANT.

3.4 Effect of Planar Preprocessing with FPP and SCHNYDER

Figures 5 and 6 show the percent improvement in each aesthetic metric when FPP and SCHNYDER are used for planar preprocessing.

For FPP, both FR and FRE had crossings reduced *significantly* for all graphs with more than 20 vertices. In particular, crossings were reduced by at least 80% for the PRUFER and R-TREES libraries, 40% for the ROME and AT&T libraries, and about 30% for the denser FUSY and EXPAN libraries, which we consider a non-trivial and clear improvement.

Preprocessing with SCHNYDER reduces crossings as well as FPP (given their similarity in Fig. 1) maintaining the 80% reduction for the trees libraries and the 30 to 40% reduction for the other four libraries. Even though SCHNYDER had a small advantage in angular resolution over using FPP when preprocessing the FUSY and EXPAN libraries, KANT also showed a similar 5-10% improvement in increasing minimum angles for those two libraries; see Fig. 3. In summary, while SCHNYDER and KANT have similar non-crossing aesthetic metrics, KANT's overall stronger performance in uniformly reducing crossings clearly demonstrates that KANT is a better choice than SCHNYDER for planar preprocessing.

4 Conclusion and Future Work

We described a set of experiments on the effect of using three standard planar embedding algorithms as a preprocessing step for four off-the-shelf force-directed algorithms. In this preliminary, proof-of-concept experiment we use all the default parameters and treat all the algorithms as black-boxes; *i.e.*, we did not attempt to adjust the parameters of one algorithm, knowing that the other algorithm will be used (or vice versa).

Surely, we could have guaranteed crossings-free drawings by using specialized algorithms designed for trees and planar graphs. But precisely because these algorithms are specialized, they are very rarely used! Instead, the vast majority of graph drawings are obtained with the default setting of popular spring embedders such as GraphViz, yEd, Gephi, etc. Adding a planar embedding preprocessing step to such existing tools has little impact on the running time for any input graph, but offers an increased likelihood of obtaining a plane layout, or reducing the crossings, if the input graph is planar.

Specifically, our results indicate that preprocessing small planar graphs reduces the number of crossings most notably for the Fruchterman-Reingold force-directed layout algorithms, while not significantly impacting any of the aesthetic metrics we measured. There was little or no effect on any of the metrics when planar preprocessing was combined with a Kamada-Kawai force-directed algorithm. There were some negative effects when planar preprocessing was combined with stress majorization. While not particularly surprising, to the best of

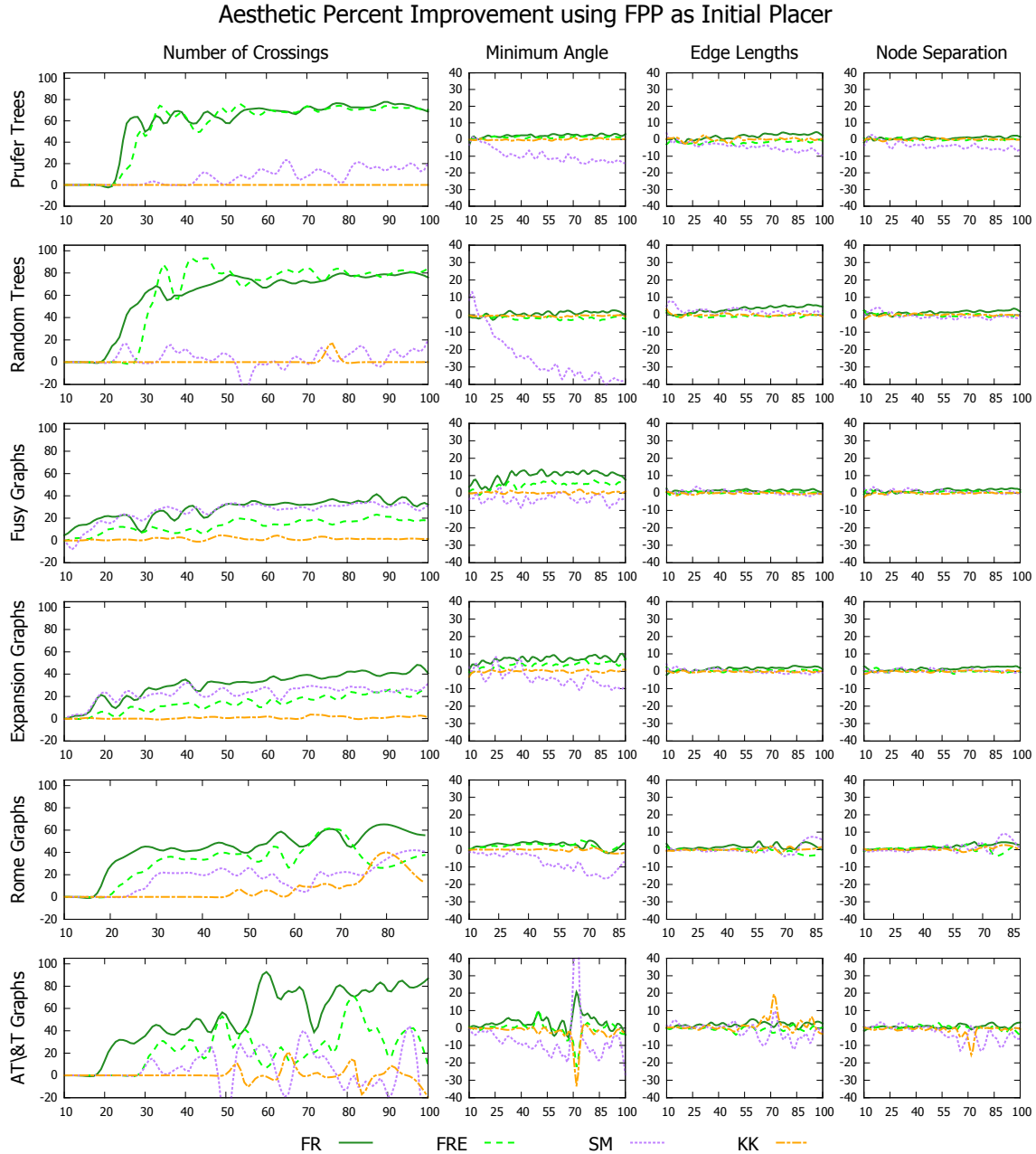


Fig. 5: Percent improvement in aesthetic metrics vs. number of nodes for spring embedders using FPP where higher, non-negative percentage is generally better.

our knowledge, these effects had not been experimentally verified before. We consider this a promising preliminary study.

The results lead to several natural follow-up questions:

1. It is very likely that integrating the preprocessing and force-directed steps will lead to better results. Specifically, is it possible to coordinate the ideal-edge-length of the spring embedder with the average edge length produced by the planar embedding? Can adjusting the cooling schedule (to ensure no big moves are made) make significant impact on the final drawing?

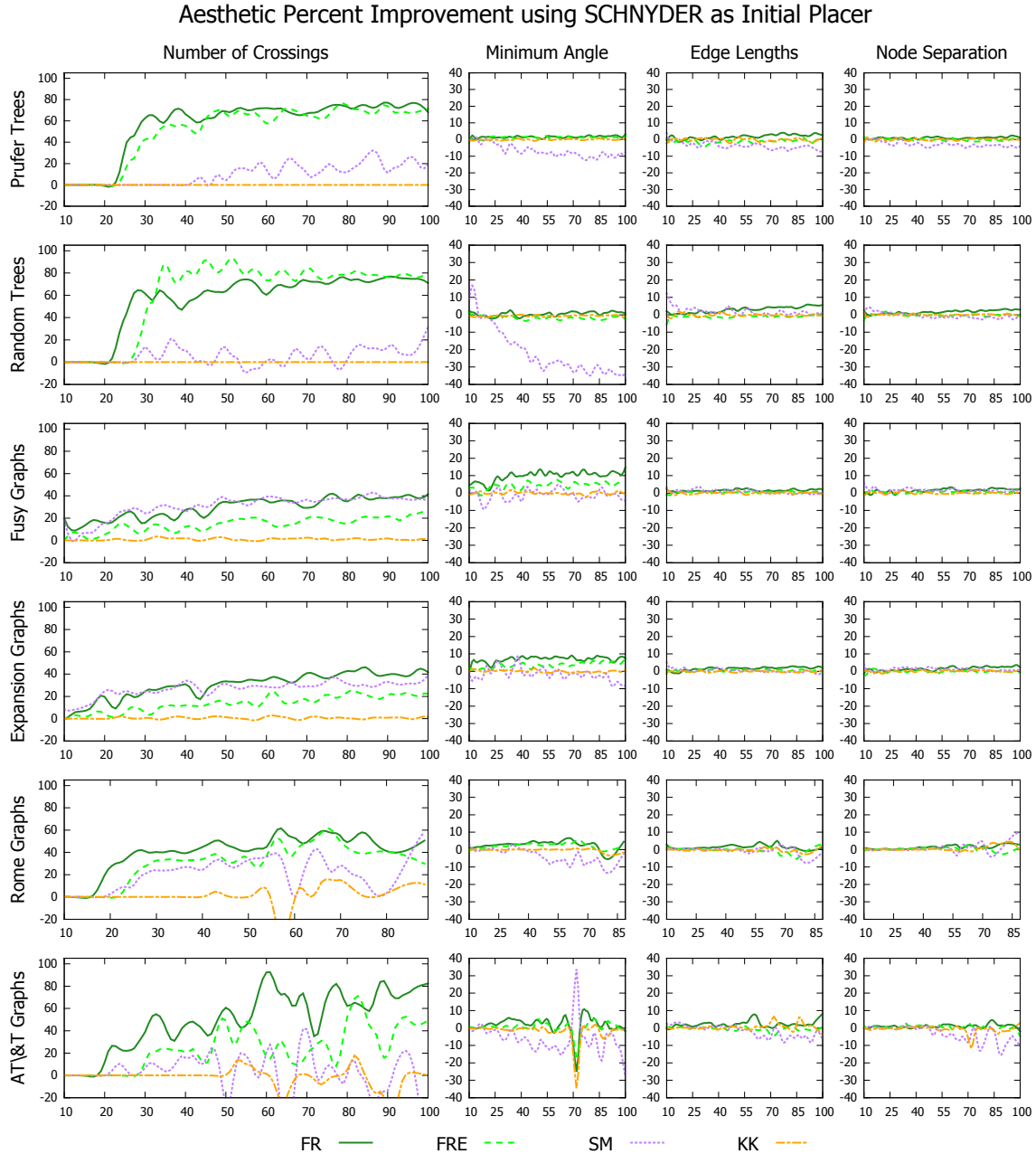


Fig. 6: Percent improvement in aesthetic metrics vs. number of nodes for spring embedders using SCHNYDER where higher, non-negative percentage is better.

2. What is the impact of planar preprocessing for larger graphs? While KANT+FRE and KK had comparable aesthetics for graphs with under 100 vertices, KK becomes impractical for larger graphs, requiring minutes to produce a layout. If the performance for large graphs remains as good, KANT+FRE might be the better option.
3. We only considered planar graphs. Is it possible to leverage planar preprocessing for non-planar graphs? For example, what is the impact on standard aesthetics when using a planar subgraph for planar preprocessing?
4. We considered several of the main aesthetics: crossings, angular resolution, node distribution, edge lengths. What other aesthetics might be worth studying when comparing the final results of spring embedder layouts?

References

1. F. Bertault. A force-directed algorithm that preserves edge-crossing properties. *Information Processing Letters*, 74(1-2):7–13, 2000.
2. F.-J. Brandenburg, M. Himsolt, and C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In F.-J. Brandenburg, editor, *Graph Drawing, Symposium on Graph Drawing, GD '95*.
3. U. Brandes and C. Pich. An experimental study on distance-based graph drawing. In *16th Symposium on Graph Drawing (GD)*, pages 218–229, 2008.
4. U. Brandes and C. Pich. More flexible radial layout. In *18th Symposium on Graph Drawing (GD)*, volume 5849, pages 107–118. 2010.
5. M. Chimani, C. Gutwenger, M. Jünger, G. Klau, K. Klein, and P. Mutzel. The open graph drawing framework. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*. CRC Press. To appear.
6. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.
7. W. Didimo, G. Liotta, and S. A. Romeo. Topology-driven force-directed algorithms. In U. Brandes and S. Cornelsen, editors, *Graph Drawing, 18th International Symposium, GD 2010*.
8. T. Dwyer, K. Marriott, F. Schreiber, P. J. Stuckey, M. Woodward, and M. Wybrow. Exploration of networks using overview-detail with constraint-based cooperative layout. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1293–1300, 2008.
9. T. Dwyer, K. Marriott, and M. Wybrow. Topology preserving constrained graph layout. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing, 16th International Symposium, GD 2008*.
10. P. Eades. A heuristic for graph drawing. *Cong. Numerantium*, 42:149–160, 1984.
11. J. Fowler and S. G. Kobourov. Planar preprocessing for spring embedders. Technical Report TR12-03, Dept. of Computer Science, Univ. of Arizona, 2012. <ftp://ftp.cs.arizona.edu/reports/2012/TR12-03.pdf>.
12. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991.
13. É. Fusy. Uniform random sampling of planar graphs in linear time. *Random Structures and Algorithms*, 35(4):464–522, 2009.
14. D. Harel and M. Sardas. Randomized graph drawing with heavy-duty preprocessing. *Journal of Visual Languages and Computing*, 6(3):233–253, 1995.
15. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
16. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996.
17. H. Prüfer. Neuer Beweis eines Satzes über Permutationen. *Archiv für Mathematik und Physik*, pages 142–144, 1918.
18. H. Purchase. Which aesthetic has the greatest effect on human understanding? In *5th Symposium on Graph Drawing (GD)*, pages 248–261, 1998.
19. H. Purchase. Metrics for graph drawing aesthetics. *Journal of Visual Languages & Computing*, 13(5):501–516, 2002.
20. W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 138–148, 1990.
21. P. Simonetto, D. Archambault, D. Auber, and R. Bourqui. ImPrEd: An improved force-directed algorithm that prevents nodes from crossing edges. *Comput. Graph. Forum*, 30(3):1071–1080, 2011.
22. D. Tunkelang. JIGGLE: Java interactive general graph layout environment. In *6th Symposium on Graph Drawing (GD)*, pages 413–422, 1998.
23. F. van Ham and M. Wattenberg. Centrality based visualization of small world graphs. *Comput. Graph. Forum*, 27(3):975–982, 2008.

5 Appendix

For purposes of comparison, we discuss the differing characteristics of our six graph libraries.

5.1 Characteristics of the Six Graph Libraries

While PRUFER and R-TREES are both tree libraries, they have different characteristics not evidenced by the statistical comparison in Table 1. The sample trees shown in Tables 3 and 4 look distinctly different. The Prüfer tree in the first table has longer chains connecting internal vertices than does the random tree seen in the second table. While Prüfer's algorithm [17]

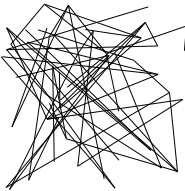
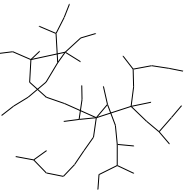
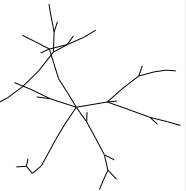
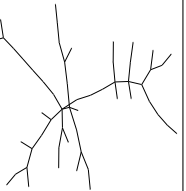
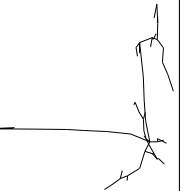
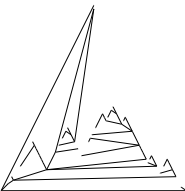
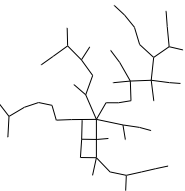
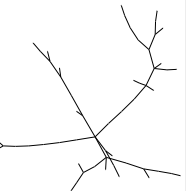
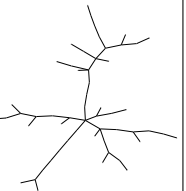
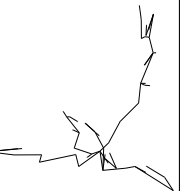
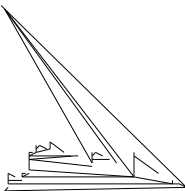
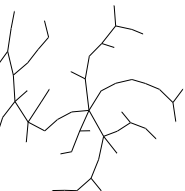

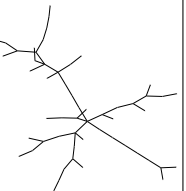

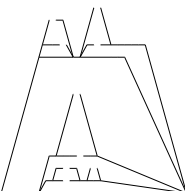
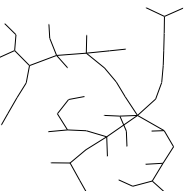
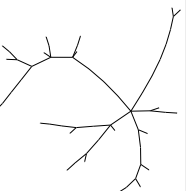
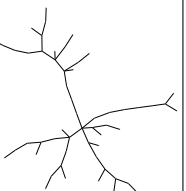
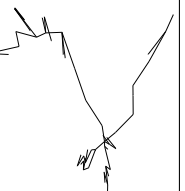
| Initial Placer | Spring Embedder | | | | |
|----------------|--|--|--|---|--|
| | None | FR | FRE | KK | SM |
| RANDOM |  c=378, ma=0.30, el=0.59, ns=0.67 |  c=4, ma=0.82, el=0.82, ns=0.83 |  c=3, ma=0.77, el=0.74, ns=0.75 |  c=1, ma=0.83, el=0.71, ns=0.79 |  c=5, ma=0.64, el=0.63, ns=0.63 |
| FPP |  c=0, ma=0.53, el=0.34, ns=0.65 |  c=1, ma=0.83, el=0.83, ns=0.82 |  c=1, ma=0.81, el=0.69, ns=0.79 |  c=0, ma=0.82, el=0.74, ns=0.79 |  c=3, ma=0.46, el=0.64, ns=0.62 |
| SCHNYDER |  c=0, ma=0.57, el=0.26, ns=0.52 |  c=0, ma=0.80, el=0.77, ns=0.85 |  c=1, ma=0.82, el=0.74, ns=0.73 |  c=2, ma=0.83, el=0.79, ns=0.79 |  c=3, ma=0.46, el=0.63, ns=0.64 |
| KANT |  c=0, ma=0.71, el=0.39, ns=0.69 |  c=1, ma=0.82, el=0.84, ns=0.82 |  c=0, ma=0.80, el=0.67, ns=0.76 |  c=0, ma=0.82, el=0.73, ns=0.80 |  c=5, ma=0.42, el=0.61, ns=0.64 |

Table 3: Layouts of a representative 55-node tree from the PRUFER library.

uniformly samples a *labeled* tree, those labels are discarded when drawn. Moreover, while there are only n non-isomorphic labelings of an n -node star (*i.e.*, $K_{1,n}$), there are $\frac{n!}{2}$ non-isomorphic labelings of an n -node path (*i.e.*, P_{n-1}). This effectively means that there exponentially many paths, or path-like trees with long chains connecting internal vertices (like the tree in Table 3), for every star or star-like tree with higher maximum degree (like the tree in Table 4).

While R-TREES is composed of random trees, they are not a uniform sampling of unlabeled trees. When iteratively choosing at random a node at each step to which to add a pendant leaf, the tree generator used to construct R-TREES does not weight by the number of auto-

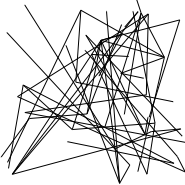
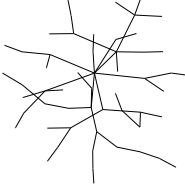
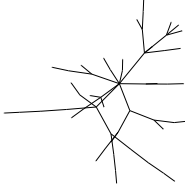
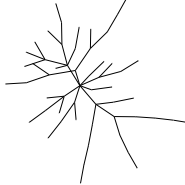
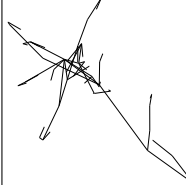
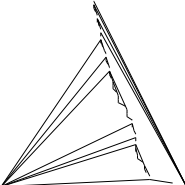
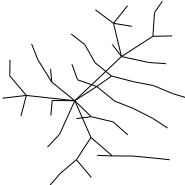
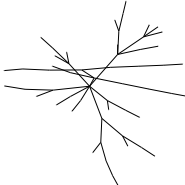
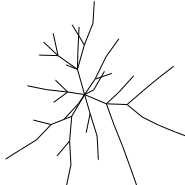
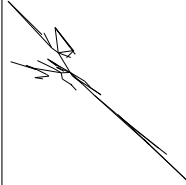
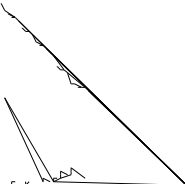

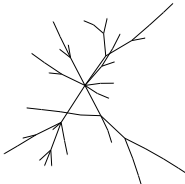
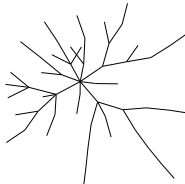
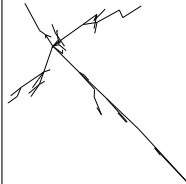
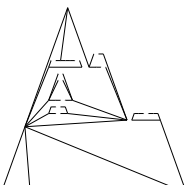
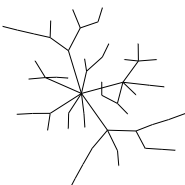
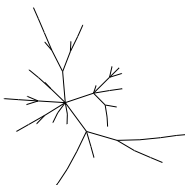
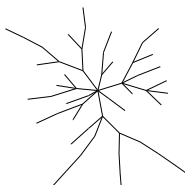
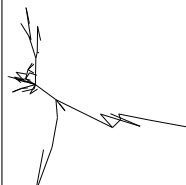
| Initial Placer | Spring Embedder | | | | |
|----------------|--|---|--|---|---|
| | None | FR | FRE | KK | SM |
| RANDOM |  c=344, ma=0.33, el=0.56, ns=0.67 |  c=8, ma=0.79, el=0.73, ns=0.84 |  c=7, ma=0.76, el=0.72, ns=0.79 |  c=2, ma=0.79, el=0.79, ns=0.77 |  c=39, ma=0.39, el=0.58, ns=0.62 |
| FPP |  c=0, ma=0.53, el=0.18, ns=0.54 |  c=3, ma=0.79, el=0.81, ns=0.87 |  c=5, ma=0.74, el=0.72, ns=0.79 |  c=2, ma=0.76, el=0.70, ns=0.78 |  c=14, ma=0.24, el=0.52, ns=0.61 |
| SCHNYDER |  c=0, ma=0.54, el=0.25, ns=0.48 |  c=10, ma=0.74, el=0.78, ns=0.84 |  c=2, ma=0.73, el=0.74, ns=0.82 |  c=1, ma=0.77, el=0.76, ns=0.80 |  c=12, ma=0.46, el=0.61, ns=0.68 |
| KANT |  c=0, ma=0.62, el=0.40, ns=0.59 |  c=1, ma=0.81, el=0.75, ns=0.84 |  c=0, ma=0.74, el=0.72, ns=0.77 |  c=0, ma=0.79, el=0.74, ns=0.80 |  c=14, ma=0.28, el=0.58, ns=0.57 |

Table 4: Layouts of a representative 55-node tree from the R-TREES library.

morphisms of subtrees to get a truly uniform sampling. For instance, when randomly adding a pendant edge to an existing k -node star, it only has a $\frac{1}{k}$ chance in adding the node to the root versus one of the leaves. However, in terms of automorphisms, either should be equally likely since there are only two possible distinct non-isomorphic trees one can construct at that point, namely a $k + 1$ -node star or the $k + 1$ -node tree obtained after subdividing an edge of a k -node star. Hence, we would expect that a library consisting of uniformly sampled unlabeled trees would have a higher maximum degree on average than is the case with either PRUFER or R-TREES.

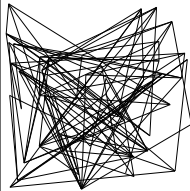
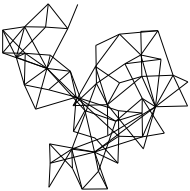

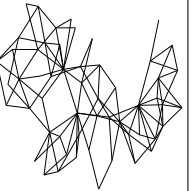
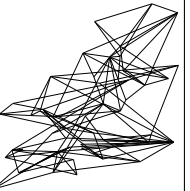
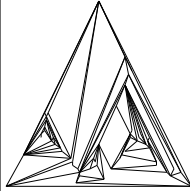
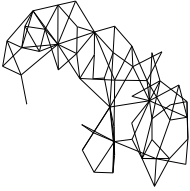
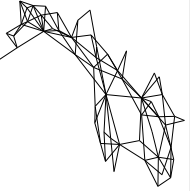
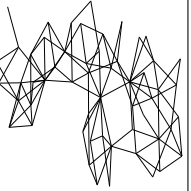
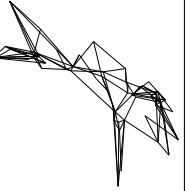
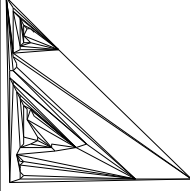
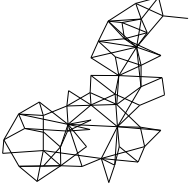
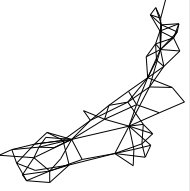
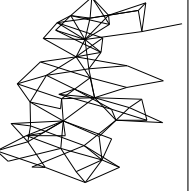
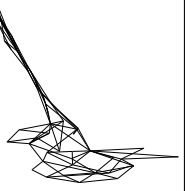
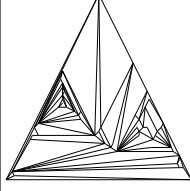
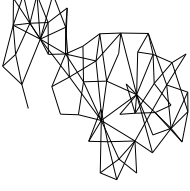
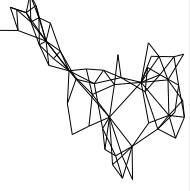
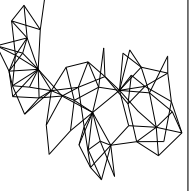
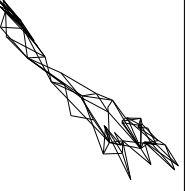
| Initial Placer | Spring Embedder | | | | |
|----------------|---|---|---|--|--|
| | None | FR | FRE | KK | SM |
| RANDOM |  c=1349, ma=0.22, el=0.67, ns=0.75 |  c=113, ma=0.41, el=0.75, ns=0.80 |  c=124, ma=0.32, el=0.67, ns=0.78 |  c=52, ma=0.40, el=0.71, ns=0.81 |  c=314, ma=0.30, el=0.64, ns=0.66 |
| FPP |  c=0, ma=0.35, el=0.50, ns=0.55 |  c=71, ma=0.46, el=0.70, ns=0.86 |  c=50, ma=0.40, el=0.69, ns=0.80 |  c=58, ma=0.37, el=0.72, ns=0.82 |  c=156, ma=0.29, el=0.61, ns=0.61 |
| SCHNYDER |  c=0, ma=0.38, el=0.44, ns=0.57 |  c=53, ma=0.45, el=0.74, ns=0.83 |  c=59, ma=0.41, el=0.75, ns=0.79 |  c=69, ma=0.40, el=0.67, ns=0.77 |  c=68, ma=0.32, el=0.71, ns=0.71 |
| KANT |  c=0, ma=0.44, el=0.48, ns=0.59 |  c=58, ma=0.42, el=0.70, ns=0.79 |  c=57, ma=0.38, el=0.71, ns=0.77 |  c=52, ma=0.43, el=0.70, ns=0.76 |  c=168, ma=0.26, el=0.66, ns=0.61 |

Table 5: Layouts of a representative 55-node graph from the FUSY library.

The graphs in FUSY were constructed with the Fusy generator referred to in [13] written in Java. Unfortunately, this is not an n -generator like the ones used to construct either R-TREES or EXPAN libraries where one only specifies the number of nodes n , and has an n -node tree or graph returned. Rather, one specifies a median value of n and then can get (with a high level of variance) graphs that deviate from n . In fact, the deviation is so huge and nonuniform, that we used the input value of $n = 1,000$ (the other two permitted input values for n are 10,000 and 100,000). We reran the generator over a million times to get 20 graphs for each value of n from 10–500. The FUSY library is a subset of that larger collection we initially generated.

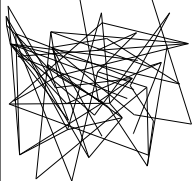
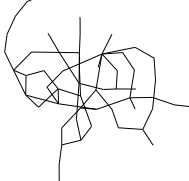
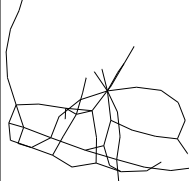
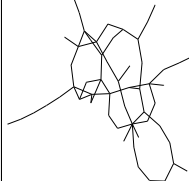
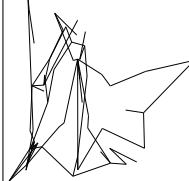
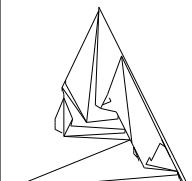
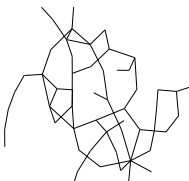
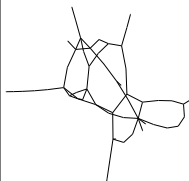
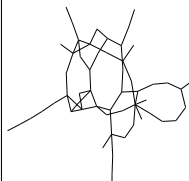
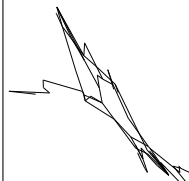
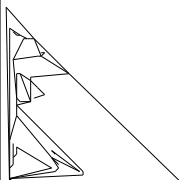
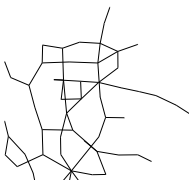
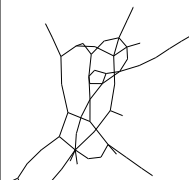
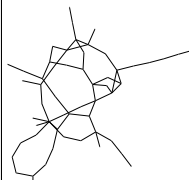
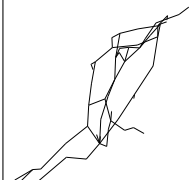
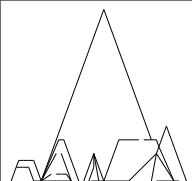
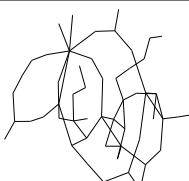
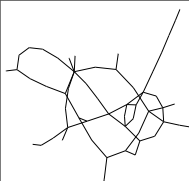
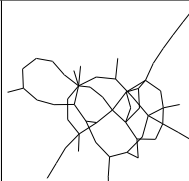
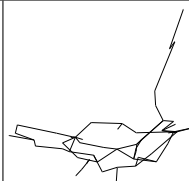
| Initial Placer | Spring Embedder | | | | |
|----------------|--|---|--|---|---|
| | None | FR | FRE | KK | SM |
| RANDOM |  c=418, ma=0.36, el=0.61, ns=0.63 |  c=18, ma=0.72, el=0.76, ns=0.84 |  c=18, ma=0.79, el=0.77, ns=0.79 |  c=7, ma=0.74, el=0.74, ns=0.81 |  c=34, ma=0.46, el=0.68, ns=0.70 |
| FPP |  c=0, ma=0.50, el=0.39, ns=0.56 |  c=9, ma=0.70, el=0.66, ns=0.83 |  c=5, ma=0.78, el=0.69, ns=0.79 |  c=7, ma=0.78, el=0.75, ns=0.81 |  c=16, ma=0.41, el=0.63, ns=0.61 |
| SCHNYDER |  c=0, ma=0.53, el=0.44, ns=0.66 |  c=10, ma=0.73, el=0.80, ns=0.84 |  c=6, ma=0.75, el=0.68, ns=0.80 |  c=6, ma=0.82, el=0.78, ns=0.84 |  c=10, ma=0.60, el=0.62, ns=0.69 |
| KANT |  c=0, ma=0.68, el=0.47, ns=0.62 |  c=10, ma=0.69, el=0.73, ns=0.81 |  c=5, ma=0.80, el=0.71, ns=0.80 |  c=7, ma=0.78, el=0.78, ns=0.81 |  c=18, ma=0.53, el=0.65, ns=0.69 |

Table 6: Layouts of a representative 55-node graph from the ROME library.

Regarding planar preprocessing, we note the most relevant distinctions between the EXPAN and the FUSY libraries; compare representative graphs from Tables 2 and 5.

1. All graphs in EXPAN are tri-connected.
2. Graphs in FUSY have a broader range of average degree than graphs in EXPAN, ranging from 2.2 to 4.8 with a median of 4.2, versus ranging from 3.6 to 5.2 with a median of 4.6 for EXPAN; see Table 1.
3. As is the case with the random n -generator used to generate R-TREES, automorphisms are not considered by the n -generator used to construct EXPAN either (i) when randomly

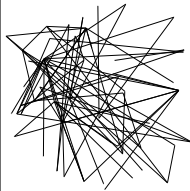
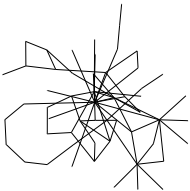
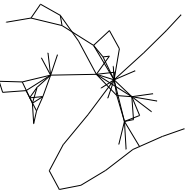
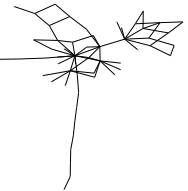
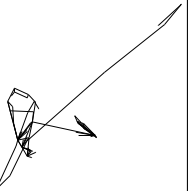
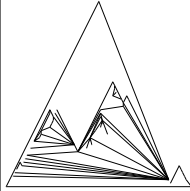
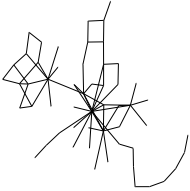
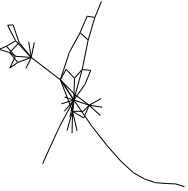
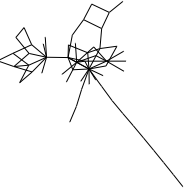
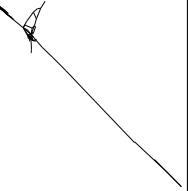
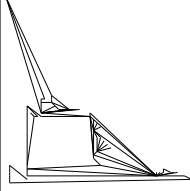
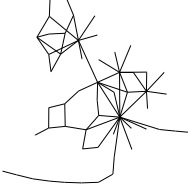
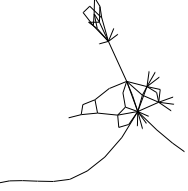
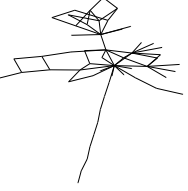
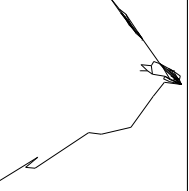
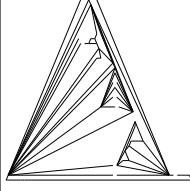
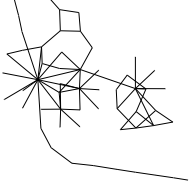
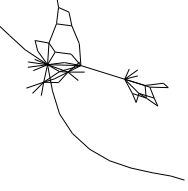
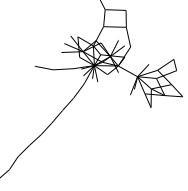
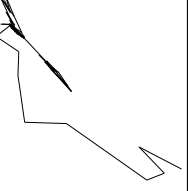
| Initial Placer | Spring Embedder | | | | |
|----------------|--|---|---|--|--|
| | None | FR | FRE | KK | SM |
| RANDOM |  c=641, ma=0.19, el=0.64, ns=0.73 |  c=81, ma=0.62, el=0.71, ns=0.78 |  c=26, ma=0.56, el=0.70, ns=0.76 |  c=8, ma=0.64, el=0.69, ns=0.76 |  c=45, ma=0.50, el=0.63, ns=0.58 |
| FPP |  c=0, ma=0.51, el=0.37, ns=0.53 |  c=20, ma=0.62, el=0.71, ns=0.84 |  c=17, ma=0.66, el=0.70, ns=0.77 |  c=24, ma=0.64, el=0.71, ns=0.75 |  c=22, ma=0.53, el=0.60, ns=0.50 |
| SCHNYDER |  c=0, ma=0.49, el=0.44, ns=0.58 |  c=4, ma=0.69, el=0.72, ns=0.86 |  c=7, ma=0.64, el=0.80, ns=0.71 |  c=10, ma=0.61, el=0.64, ns=0.75 |  c=31, ma=0.41, el=0.63, ns=0.48 |
| KANT |  c=0, ma=0.55, el=0.35, ns=0.62 |  c=17, ma=0.66, el=0.76, ns=0.82 |  c=11, ma=0.67, el=0.80, ns=0.68 |  c=22, ma=0.64, el=0.72, ns=0.76 |  c=104, ma=0.31, el=0.62, ns=0.44 |

Table 7: Layouts of a representative 55-node tree from the AT&T library.

selecting nodes to split or (ii) when randomly distributing the neighbors of the split node (while keeping the graph triconnected).

4. All three of the planar preprocessors must first augment a graph until it becomes biconnected, where any added edges are then discarded at the end. However, no augmentation is required for graphs in **EXPAN** when preprocessing. This is possibly one reason to account for **EXPAN** having slightly greater crossing reduction on average than **FUSY** has for each of the three preprocessors.

Finally, graphs seen in Tables 6 and 7 are fairly representative of the 28% of the **Rome** graphs and the 67% of the **AT&T** graphs, which are planar, that compose the **ROME** and the **AT&T** libraries, respectively. Most of the graphs from either library have low average degree ranging from 1.8 (trees) to 3.8 with a median of 2.3 for the **ROME** library and ranging from 1.8 to 4.7 with a median of 2.2 for the **AT&T** library; see Table 1. Additionally, each had a much lower median number of nodes per graph of 23 or 24 nodes versus 55 nodes as is the case with the other four libraries.