Drawing planar cubic 3-connected graphs with few segments : algorithms & experiments

Drawing Planar Cubic 3-Connected Graphs with Few Segments: Algorithms & Experiments*

Alexander Igamberdiev[1], Wouter Meulemans[2], and André Schulz[1]

[1] LG Theoretische Informatik, FernUniversität in Hagen, Germany
[alexander.igamberdiev|andre.schulz]@fernuni-hagen.de
[2] giCentre, City University London, United Kingdom
wouter.meulemans@city.ac.uk

Abstract. A drawing of a graph can be understood as an arrangement of geometric objects. In the most natural setting the arrangement is formed by straight-line segments. Every cubic planar 3-connected graph with n vertices has such a drawing with only $n/2+3$ segments, matching the lower bound. This result is due to Mondal et al. [J. of Comb. Opt., 25], who gave an algorithm for constructing such drawings.
We introduce two new algorithms that also produce drawings with $n/2+3$ segments. One algorithm is based on a sequence of dual edge contractions, the other is based on a recursion of nested cycles. We also show a flaw in the algorithm of Mondal et al. and present a fix for it. We then compare the performance of these three algorithms by measuring angular resolution, edge length and face aspect ratio of the constructed drawings. We observe that the corrected algorithm of Mondal et al. mostly outperforms the other algorithms, especially in terms of angular resolution. However, the new algorithms perform better in terms of edge length and minimal face aspect ratio.

1 Introduction

To assess the quality of network visualizations, many criteria have been investigated, such as crossing minimization, bend minimization and angular resolution (see [9] for an overview). The structural complexity of a graph is often measured in terms of its number of vertices or edges. This, however, does not necessarily correspond to its *cognitive load* (mental effort needed to interpret a drawing). Bends and crossings increase the cognitive load, making it harder to interpret a graph visualization, and should be avoided.

We consider the following measure of *visual complexity* for planar graphs [8]: the number of basic geometric objects that are needed to realize the drawing. For example, if a path in the graph is placed along a line, then we do not need one line segment for each edge in this path; one line segment can represent the entire path. In contrast to bends and crossings, which *increase* the cognitive load, this definition of visual complexity aims to measure a *reduction* in cognitive

* Funded by the German Research Foundation (DFG) under grant SCHU 2458/4-1.

load in comparison to the structural complexity. The basic geometric shapes are typically straight-line segments or circular arcs. Upper and lower bounds on the necessary visual complexity of various graph classes are known [2,3,8]. A lower bound for any graph is $N/2$, where N is the number of odd-degree vertices: at least one geometric object must have its endpoint at such a vertex. Computing the optimal visual complexity of line-segment drawings is NP-hard [4].

We consider line-segment drawings for *planar cubic 3-connected graphs*; unless mentioned otherwise, "graph" is used to refer to a graph of this class. Any plane drawing has at least three vertices of the same face on its convex hull: such a vertex is the endpoint of the line segment for each incident edge. Thus, we obtain a lower bound of $n/2+3$ line segments, as n is even. Dujmović gave an algorithm for drawing general planar graphs with low visual complexity [2]. This algorithm will draw a cubic planar graph with $n+2$ line segments. Mondal et al. [7] improve this by giving an algorithm that uses $n/2+4$ segments. Moreover this algorithms places the vertices on a $(n/2+1) \times (n/2+1)$ grid and uses only 6 different slopes. A variant of the algorithm is also suggested, one that does not place the vertices on a grid and uses 7 distinct slopes, but attains a visual complexity of $n/2+3$. The presentation of Mondal et al. contains a flaw, but it can be fixed as discussed in Sect. 4.

To compute a plane drawing matching the lower bound, we are given (or pick) three convex hull vertices; these are referred to as the *suspension vertices*. For all other *internal vertices*, we decide which two incident edges lie on the same line segment, that is, which of the three angles is flat. Hence, this corresponds to a *flat-angle assignment*; we refer to plane drawings that match the lower bound as *flat-angle drawings*. Note that any face in a flat-angle drawing is nonstrictly convex. Aerts and Felsner [1] describe conditions for the stretchability of flat-angle assignments. From a stretchable assignment, a layout can be obtained by solving a system of harmonic (linear) equations with arbitrary edge weights, very similar to the directed version of Tutte's barycentric embedding as presented by Haas et al. [6]. How to efficiently compute stretchable flat-angle assignments remains an open problem.

Contributions. We present two different new $O(n^2)$-time algorithms (Sect. 2 and 3) to construct a plane drawing with $n/2+3$ segments for $n \geq 6$, matching the lower bound. From the constructed drawings, a flat-angle assignment is derived, which is then used to set up a system of harmonic equations [1]. By solving the system using uniform edge weights we can redraw the layouts to (possibly) increase their visual appeal. To the best of our knowledge the new algorithms present novel methods to incrementally build up cubic planar 3-connected graphs by simple and local modifications. These construction sequences might also find applications outside of our applications.

We review the algorithm of Mondal et al. and discuss cases where it might produce degenerate drawings. We then present a fix for these problematic cases. This leaves us with three algorithms that produce drawings of cubic planar 3-connected graphs with low visual complexity; see Fig. 1. We run several experiments and evaluate the performance of these algorithms by measuring geometric

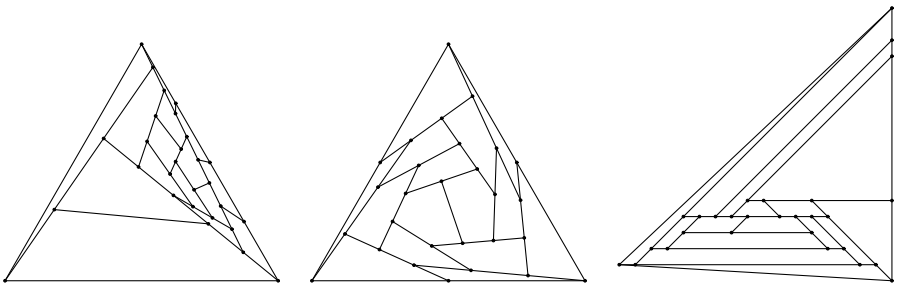

Fig. 1. Result of the various algorithms for the same graph and outer face. (left) Deconstruction algorithm. (middle) Windmill algorithm. (right) Mondal algorithm.

features of the produced drawings. In particular, we measure angular resolution, edge length, and face aspect ratio. We use two data sets for our experiments. For the first data set we sample over the set of all cubic planar 3-connected graphs with 24 to 30 vertices. The second data set is given by the set of 146 popular graphs with at most 30 vertices from the Wolfram graph database[3].

2 The Deconstruction Algorithm

For this algorithm we define an operation called *edge insertion*[4]: pick two edges that belong to one face, subdivide both edges and add a new edge between the new degree-2 vertices while preserving planarity. It is folklore that every cubic 3-connected graph can be obtained from K_4 by a sequence of edge insertions (e.g, [5, page 243]). For our purpose we need a slightly stronger version (proven in the full version): any cubic planar 3-connected graph other than K_4 can be constructed by a sequence of edge insertions from the triangular prism, while not adding new edges in a given outer face (though outer-face edges may be subdivided).

An edge whose removal (understood as a reverse edge insertion) maintains planarity, 3-regularity, 3-connectivity, and a chosen outer face is called a *good edge*. We compute a construction order by repeatedly removing a good edge (a good edge always exists as proven in the full version). This procedure always finishes on a triangular prism which has a trivial flat-angle drawing for any outer face. Note that the construction sequence is not necessarily unique. For the later analysis (Sect. 5), we distinguish three different strategies on how to select the removed edge from the set of all good edges in every step:

(R) We select the edge randomly from the set of all good edges.
(S) We select a good edge with minimal sum of the degrees of its incident faces.
(L) Analogous to (S), we select a good edge with maximal sum of degrees.

[3] `http://reference.wolfram.com/language/ref/GraphData.html`
[4] This is *not* the graph-theoretic notion of edge insertion.

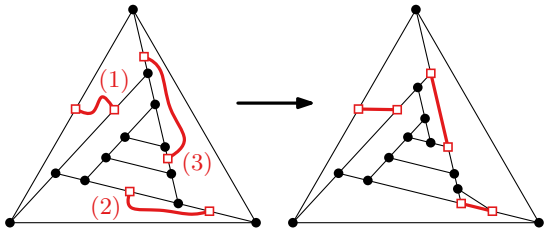

Fig. 2. Edge insertion that connects (1) noncollinear edges of a face, (2) collinear edges separated by one vertex, and (3) collinear edges separated by two or more vertices. In case 2 and 3, we reassign the flat angle at the first and/or last separating vertex.

We remark that there is no basis to suggest that the strategies (S) or (L) might perform particularly well: we study these strategies primarily to have a more structured procedure against which we can compare the randomized strategy.

Once we have obtained the construction order, we can reconstruct the original graph from the triangular prism with a sequence of edge insertions maintaining a flat-angle drawing (see Fig. 2). When inserting edges, we may have different possibilities how to update the flat-angle assignment. Depending on our strategy we may obtain different drawings. If an edge insertion "connects" two edges that are not aligned, we have an obvious way how to add the new edge: we pick a subdivision point on each of the edges and add the new edge as a straight-line segment connecting these points. If the two edges are aligned (part of a common segment ℓ), we need to modify the existing drawing. Let u and v be the new vertices that we introduce and let $s_1, \ldots, s_k$ be the vertices in between u and v on ℓ; see Fig. 3(a). Since the graph after adding e is planar, all segments starting at s_i have to leave ℓ on the same side. We first draw the new edge (u, v) on top of ℓ such that v coincides with s_k. To repair the degeneracy, we tilt the old part of ℓ that was running between u and s_k as done in Fig. 3(b). Here we let s_k "slide" on its segment that was not part of ℓ. For $k \geq 2$ we have also the following alternative how to insert (u, v): we draw a segment parallel to ℓ that runs between the segments starting at s_1 and s_k. We place all endpoints s_i on this new segment without changing any slopes of the old segments. We now take the old vertex s_1 as u, and the old vertex s_k as v as depicted in Fig. 3(c). We refer to the latter strategy as the *alternate insertion operation*.

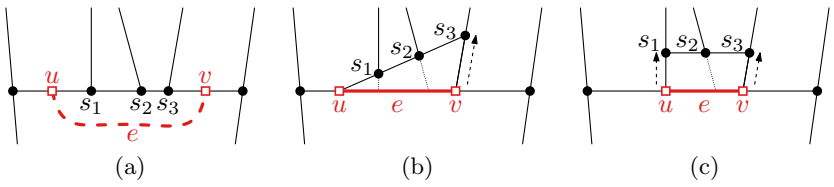

Fig. 3. (a) Inserting edge $e = (u, v)$ with its endpoints on the same side of a face (a). (b) The standard insertion. (c) An alternative strategy.

With three different strategies (R, S and L) and an alternative insertion operation (ALT), we have six variants of the Deconstruction algorithm, referred to as DEC-R, DEC-R-ALT, DEC-S, DEC-S-ALT, DEC-L and DEC-L-ALT.

3 The Windmill Algorithm

The Windmill algorithm computes a flat-angle drawing, working its way inward from the outer face, until all vertices have been processed. It does so recursively, using as parameter a simple cycle C in the graph. It assumes that C is drawn as a nonstrictly convex polygon. Its convex corners correspond to suspension vertices or vertices having an edge outside C; any flat vertex has an edge inside C. Initially, C is the outer face, drawn as an equilateral triangle with the suspension vertices as corners (Fig. 4(a)). Based on the cyclic sequence F of faces along the inside of C, a recursive step for cycle C is done using the first of the cases below:

1. If at most one vertex lies inside C, we draw all chords as line segments. The one vertex (if present) is positioned to lie on a line segment between two of its neighbors. See Fig. 4(b→c,e→f).
2. If a face occurs more than once in F, we draw its paths inside C as line segments and recurse on a subcycle for each path. See Fig. 4(c→d).
3. If two faces share an edge, but are not consecutive in F, we draw three line segments to represent the paths inside C along the two faces and recurse on the two subcycles created. See Fig. 4(a→b).
4. Otherwise, we create a windmill pattern with the sequence of faces along C. We recurse on the cycle inside the windmill. See Fig. 4(d→e).

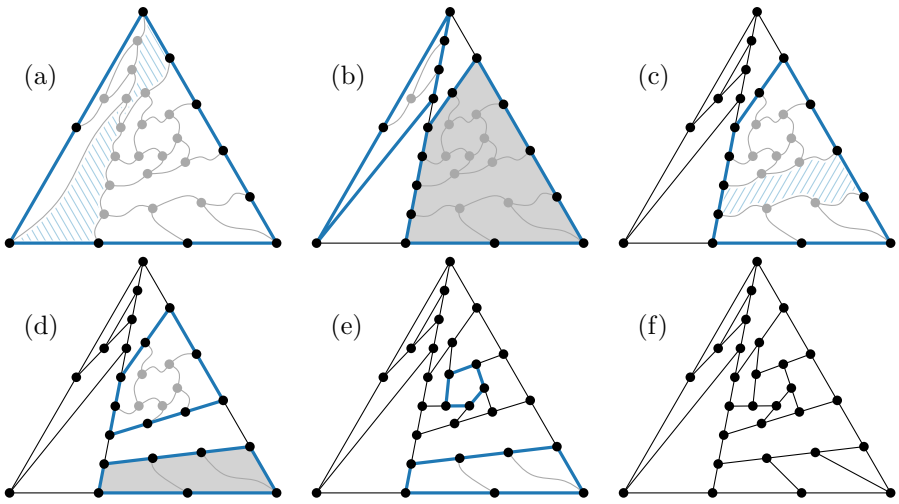

Fig. 4. The Windmill algorithm. Cycles are drawn thick; unshaded cycles are processed in the next step. (a) Initial call. (b→ . . . →e) Consecutive states. (f) Final result. (e→f) Two cycles are processed. (a,c) Hashures indicate faces relevant for case 3 and 2.

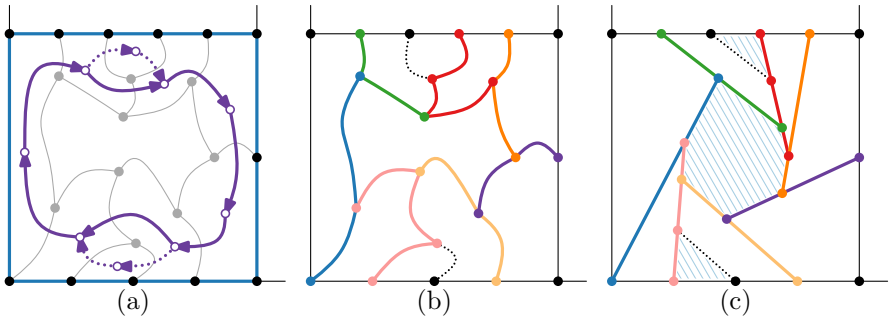

Fig. 5. (a) The dual restricted to F. Shortest cycle is given with solid lines. (b) Decomposing the face boundaries for the windmill structure. (c) Drawing the windmill pattern and the bypassed components as triangles. Three cycles are recursed on (hashures).

There is a subtlety for case 3: the faces must lie on different sides of the polygon for C. Otherwise, the condition on C described above cannot be maintained in a plane drawing. Therefore, we use case 4 to handle such a pattern using additional recursive calls. Below, we sketch how this works.

To construct a windmill, we proceed as follows (see Fig. 5). First, we consider the dual of the given graph, restricted to the vertices that are dual to the faces in F. Ideally, this is a simple cycle. However, two nonconsecutive faces of F that are adjacent (and, because of case 3, lie on the same side of C) cause a chord in this cycle. We find the shortest cycle in this restricted dual to create the windmill (Fig. 5(a)). The face boundaries of the faces on this cycle are decomposed as to provide the basic windmill structure (Fig. 5(b)). This bypasses any components separated by a chord in the dual. These are inserted as triangles at the correct place and recursed on as well, after drawing the windmill pattern (Fig. 5(c)).

Windmills can be created in a clockwise or counterclockwise direction. To decide, we provide two strategies. The first is to always choose the same direction; the other is to alternate clockwise and counterclockwise, depending on the recursion depth. We refer to these variants as WIN and WIN-ALT respectively.

Crucial to the proof of correctness is showing that any cycle C is nonstrictly convex and any vertex on C, for which the third edge is not drawn yet, is either a suspension vertex or its other two edges (part of C) are drawn collinearly: in either case, we need not worry about aligning the undrawn edge with another to obtain minimal visual complexity (since each nonsuspension vertex must have exactly two aligned edges). For full details and proofs, we refer to the full version.

4 The Mondal Algorithm

Mondal et al. [7] describe two linear-time algorithms for drawing cubic planar 3-connected graphs: one results in a grid drawing with $n/2+4$ line segments; the other attains minimal visual complexity but does not produce a grid drawing. Both algorithms introduce the vertices as given by a canonical order.

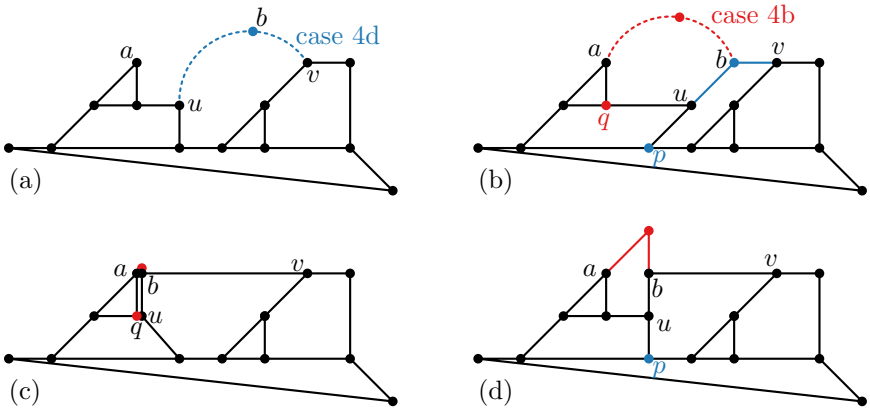

Fig. 6. (a) State before adding b between u and v. (b) After adding b, before adding a vertex between a and b. (c) Rotating about q as described in [7] results in a nonplane drawing. (d) Rotating about p as described here yields a plane result.

We observed that the grid algorithm as described by Mondal et al. [7] is flawed. The example in Fig. 6 illustrates the problem. When adding a chain of vertices from u to v (case 4d in [7], see Fig. 6(a-b)), the vertex at u is "rotated" to give it an incident edge with slope 1. In the next step, we may need to rotate backward to give vertex b an incident edge with slope ∞ (case 4b in [7], see Fig. 6(b-c)). However, the point computed to rotate about is erroneous: it is point q. This causes u to be placed on top of q, resulting in a nonplane drawing.

To resolve this issue, we suggest the following procedure for determining the correct pivot point. For case 4b, we walk downward along the slope 1 edges, until we find a pivot vertex p that has either two slope 0 edges or a downward edge with slope 1 and downward edge with slope ∞. In this case, every vertex w along the path is moved $\ell(w)$ positions to the left, where $\ell(w)$ is the vertical distance between w and pivot vertex p. We refer to this as a *left-rotation*. The correct result for the counter example is given in Fig. 6(d).

Analogously for case 4d, we walk downward along the slope ∞ edges, until we find a pivot vertex p that has either two slope 0 edges or a downward edge with slope 1 and downward edge with slope ∞. In this case, every vertex w along the path is moved $r(w)$ positions to the right, where $r(w)$ is the vertical distance between w and pivot vertex p. We refer to this as a *right-rotation*.

To prove that left- and right-rotations maintain a plane drawing, we must show that for every degree-3 vertex along the path to the pivot vertex, any horizontal edge in the direction of the rotation has sufficient length. This is captured by the invariant below. To simplify notation, we define $r(w)$ and $\ell(w)$ to be 0, if w is not on a path that may be right-rotated or left-rotated respectively.

Invariant 1 *Consider an edge $e = (u, v)$ with slope 0 and let u and v be its left and right vertex respectively. The length of e is at least $1 + r(u) + \ell(v)$.*

Observe that $r(u)$ or $\ell(v)$ is nonzero only in situations where u has been left-rotated or v has been right-rotated. To fully prove this statement is out of scope for this paper. Also, note that this is the invariant for the grid algorithm. For the minimal-complexity algorithm, we must multiply the values of $r(\cdot)$ and $\ell(\cdot)$ by two, and observe that rotations are not performed with slope ∞ edges: their role is taken by slope -1 edges.

Moreover, we observe that the Mondal algorithm achieving minimal visual complexity can be easily adapted to lie fully on a grid and use only six slopes as well. To this end, we need to do only the following: whenever the bottom point is moved to the right, it is moved downwards for an equal distance. This ensures that its incident edge maintains a slope of -1.

Thus, we have two variants of the Mondal algorithms, both on a grid and with only 6 slopes for its edges: one uses $n/2 + 4$ line segments, but draws on a smaller grid than the second algorithm that uses only $n/2 + 3$ line segments. We refer to these as MON-GRID and MON-MIN respectively.

5 Experiments

We have three different algorithms (each with its own variants) to draw planar cubic 3-connected graphs using only $n/2+3$ line segments. The drawings (Fig. 1) are obviously different, but—as the visual complexity is the same—we need criteria to further assess the overall quality. In this section we discuss experimental results comparing the 10 algorithm variants described in the previous sections.

5.1 Graphs

We generated all planar cubic 3-connected graphs with 24, 26, 28 and 30 vertices, using *plantri*[5]. From each batch we sampled 500 graphs uniformly at random, resulting in a total of 2000 graphs. The Wolfram data set shows roughly the same patterns can be observed as for the random data set. As the graphs in this data set are typically smaller, some differences arise.

5.2 Measures

We use the following three measures to quantify the quality of a graph layout.

Angular resolution. At each internal vertex in the graph, we measure the smallest angle as an indicator of angular resolution. Since one angle is always π, the best angular resolution is $\pi/2$. Angular resolution measures how easily discernible the incident edges are. A high value indicates a good angular resolution.

Edge length. We measure all edge lengths in the graph, normalized to a percentage of the diagonal of the smallest enclosing axis-aligned square. Though

[5] `http://cs.anu.edu.au/~bdm/plantri/`

edge lengths should neither be too short nor too long, we in particular look at avoiding long edges[6]: we consider lower values for edge length to be better.

Aspect ratio. For each face, we measure the aspect ratio of the smallest enclosing (not necessarily axis-aligned) rectangle. To compute this ratio, we divide the length of its shorter side by the length of its longer side, yielding a value between 0 and 1. High values thus indicate a good aspect ratio. This is a simple indicator of fatness, as all faces are convex.

Measuring procedure. For each graph, we run each algorithm using each possible face of the graph as an outer face. For each measure, we compute both the average value over all elements (vertices, edges, faces) as well as the worst value. The worst value is the minimum value for angular resolution and face aspect ratio, and the maximum value for the edge length. For both the average and worst value, we compute the average over all drawings for a particular graph, i.e., what may be expected for that graph if we had chosen an outer face uniformly at random. Thus, we have six measures in total.

5.3 Algorithm comparison

Fig. 7 shows the measured results for all graphs in the data set, summarized as a box plot. For the DEC algorithms, only the ALT variants are shown, as the results of the other variants are very similar.

Angular resolution. The MON algorithms clearly perform better than the WIN algorithms, which in turn outperform the DEC algorithms. This was to be expected due to the fixed slopes used in the MON algorithms. We observed that for the Wolfram data set, the angular resolution tends to increase for the WIN and DEC algorithms. However, for the MON algorithms, there in fact seems to be a slight decrease.

Edge length. The worst-case values show that the MON algorithms perform worst, and the WIN algorithms perform best; average edge length shows that MON is slightly behind the WIN and DEC algorithms. Though statistically significant (later in this section), the differences are only minor. The maximum edge length for the WIN and DEC algorithms is lower due to its placement in an equilateral triangle and the possibility of having additional vertices on all sides of this triangle; the MON algorithms always have a long edge, close to the diagonal of the drawing. For the MON-MIN algorithm, this worst-case edge length is smaller than for MON-GRID. This is caused by our modification which moves one point downward, thereby increasing the grid size.

Face aspect ratio. We see that the DEC algorithms are outperformed by the other algorithms in terms of average ratio. MON-MIN outperforms MON-GRID and the WIN algorithms. However, looking at the minimal face aspect ratio of a drawing, we see that DEC outperforms the MON algorithms, and MON-GRID is actually slightly ahead of MON-MIN. For the Wolfram data set, containing

[6] Informal investigation of minimal edge lengths suggested only tiny differences, though MON-GRID was slightly ahead of the other algorithms.

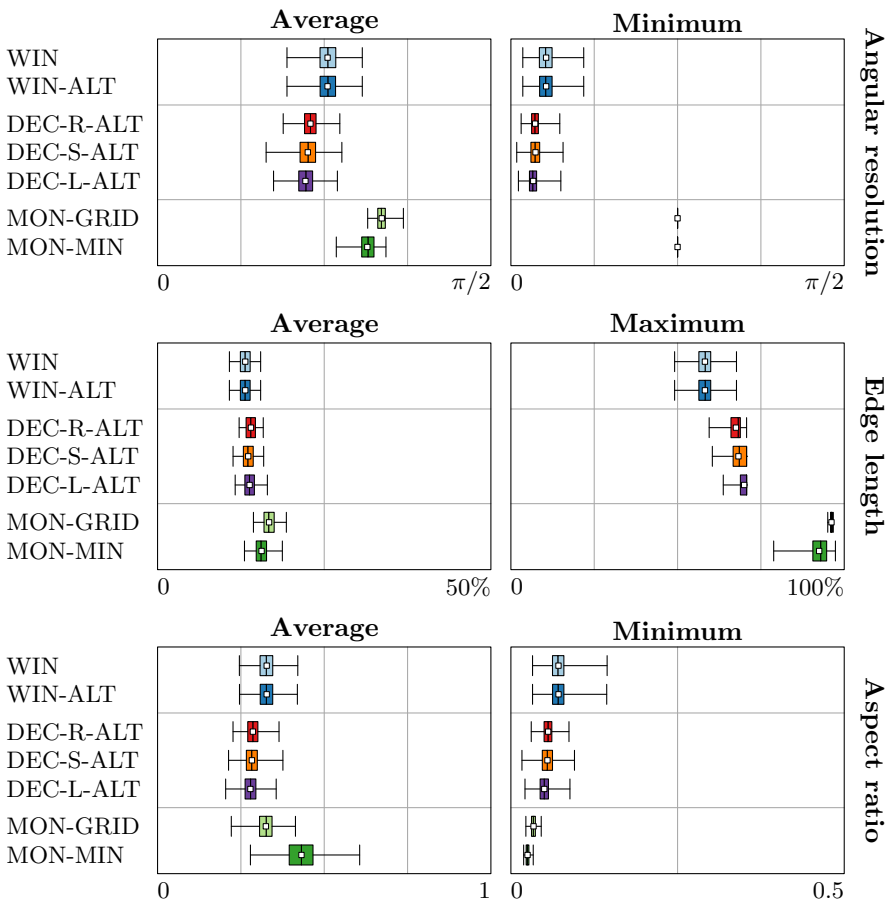

Fig. 7. Box plot of the measured results. For length, lower values indicate better drawings; for the other measurements, higher values indicate better drawings.

smaller graphs ($n = 16.5$ on average), we observe that the average face aspect ratio of the MON algorithms decreases: MON-MIN is in line with the DEC algorithms and the MON-GRID algorithm has lost its lead on the WIN algorithms.

We conclude from the above that the WIN algorithms generally outperform DEC algorithms. Between the WIN and MON algorithms, there is no clear agreement between the measures: the MON algorithms perform very well in angular resolution, but worse in edge length; for the face aspect ratio, it depends whether we consider the average or minimum ratio in a drawing.

Statistical analysis. We further investigate the differences by performing an RM-ANOVA analysis on the measurements with a post-hoc Tukey HSD test to reveal the pairwise differences. The Skewness and Kurtosis of all measurements are within the range $[-2, 2]$, thus providing evidence for the assumption of the normal distribution for these analyses. The only exception is the minimal an-

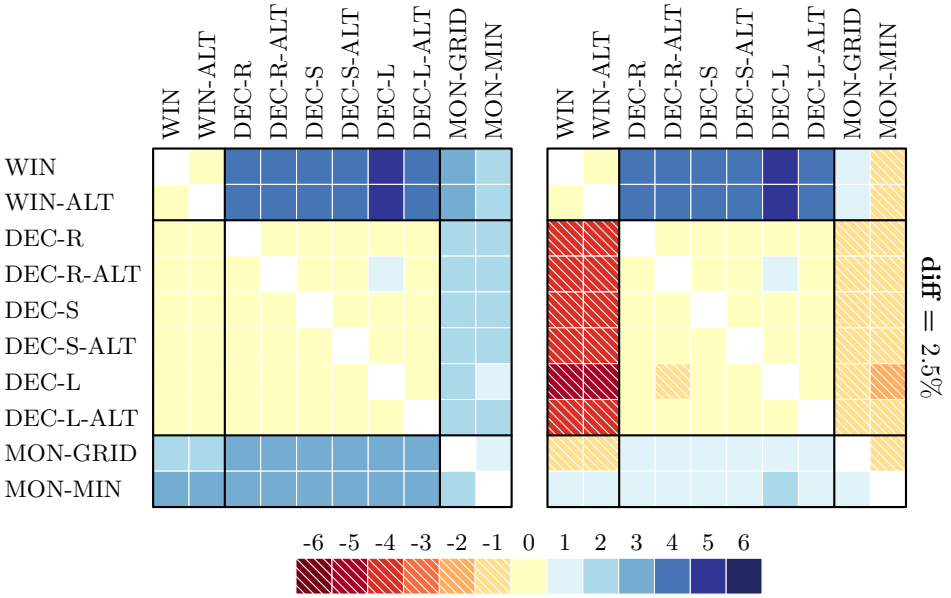

Fig. 8. (Left) Number of "wins" (measures for which the row-algorithm outperforms the column-algorithm), using $p < 0.001$ in a Tukey HSD test with an estimated difference in means of at least 2.5%. (Right) The number of "wins" minus the number of "losses" in the left table, giving an overall view of relative performance.

gular resolution for the MON algorithms, which are constant at $\pi/4$. These are excluded from the statistical analysis; we consider the MON algorithms to (significantly) outperform the other algorithms due to the high difference in means.

The results of this statistical investigation are summarized in Fig. 8. For this, we require an estimated difference in means of at least 2.5% of the possible range of values, i.e., a difference of $\pi/80$ for angular resolution, 2.5% for edge length and 0.025 for face aspect ratio.

We verify that the WIN algorithms clearly outperform the DEC algorithms, in at least four measures (out of six). Between the two variants, there is no difference. As observed above, whether the MON algorithms outperform another algorithm depends highly on the measure. The MON-MIN algorithm "wins" more often than it "loses" compared to any other algorithm. However, the MON-GRID algorithm is outperformed by the WIN algorithms. The DEC algorithms are not distinguishable between themselves. They outperform the MON algorithms for some measures, but are outperformed by the MON algorithms more often.

6 Conclusions

We studied algorithms for drawing cubic planar 3-connected graphs with minimal visual complexity, i.e., with as few line segments as possible. The lower bound

is $n/2+3$ for a graph with n vertices, and we introduced two new algorithms to match this lower bound. These algorithms may be of independent interest, as a way of constructing planar cubic 3-connected graphs. Moreover, we resolved a flaw in an existing algorithm by Mondal et al. [7].

This leaves us with three algorithms, each with two or more variants. We performed an experiment with two data sets to compare the performance of these algorithms in terms of angular resolution, edge length and face aspect ratio. The Deconstruction algorithm is always outperformed by the Windmill algorithm, but the Windmill algorithm seems to be on par with the Mondal algorithm: depending on the criterion, one or the other performs better. One aspect that was not taken into consideration though, is that the Mondal algorithm comes with a maximum grid size and uses only 6 slopes to draw the line segments.

Future work. We studied visual complexity for planar cubic 3-connected graphs, which is rather restrictive. Future algorithmic work may aim towards reducing the gap between upper and lower bounds for other graph classes such as triangulations or general planar graphs (see [3]). Moreover, the definition of visual complexity is not limited to line segments, but may include for example the use of circular arcs (see [8]). We may investigate how many vertices are spanned by a line segment—but what is "better" here is not immediately clear. Moreover, we may look into applying the system of harmonic equations to the Mondal layouts.

Furthermore, it would be interesting to investigate whether the definition of visual complexity correlates to an observer's assessment of complexity. In other words, are drawings with minimal visual complexity indeed perceived to be simpler than those with higher visual complexity? Moreover, can we establish a relation between visual complexity and cognitive load? The graph may be visually simpler, but that does not readily imply that it is easier to interpret.

References

1. N. Aerts and S. Felsner. Straight line triangle representations. In S. Wismath and A. Wolff, editors, *Graph Drawing*, LNCS 8242, pages 119–130, 2013.
2. V. Dujmović, D. Eppstein, M. Suderman, and D. Wood. Drawings of planar graphs with few slopes and segments. *CGTA*, 38:194–212, 2007.
3. S. Durocher and D. Mondal. Drawing plane triangulations with few segments. In *Proc. of 26th CCCG*, pages 40–45, 2014.
4. S. Durocher, D. Mondal, R. Nishat, and S. Whitesides. A note on minimum-segment drawings of planar graphs. *J. of Graph Alg. and Appl.*, 17(3):301–328, 2013.
5. B. Grünbaum. *Convex Polytopes*. Graduate Texts in Math. Springer-Verlag, New York, 2nd edition, 2003.
6. R. Haas, D. Orden, G. Rote, F. Santos, B. Servatius, H. Servatius, D. L. Souvaine, I. Streinu, and W. Whiteley. Planar minimally rigid graphs and pseudo-triangulations. *Comput. Geom.*, 31(1–2):31–61, 2005.
7. D. Mondal, R. Nishat, S. Biswas, and S. Rahman. Minimum-segment convex drawings of 3-connected cubic plane graphs. *J. of Comb. Opt.*, 25:460–480, 2013.
8. A. Schulz. Drawing graphs with few arcs. In A. Brandstädt, K. Jansen, and R. Reischuk, editors, *Graph-Theoretic Concepts in CS*, LNCS 8165, pages 406–417, 2013.
9. R. Tamassia. *Handbook of Graph Drawing and Visualization*. CRC Press, 2013.