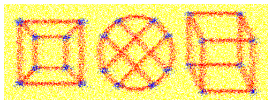
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 21, no. 4, pp. 561–588 (2017)
DOI: 10.7155/jgaa.00430

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

Alexander Igamberdiev *Wouter Meulemans*[1] *André Schulz*[2]

[1]Dept. of Mathematics and Computer Science, Eindhoven University of Technology, the Netherlands
[2]LG Theoretische Informatik, FernUniversität in Hagen, Germany

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.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
October 2016	December 2016	February 2017	April 2017	April 2017
		Published: April 2017		
	Article type: Regular paper		Communicated by: G. Liotta	

Funded by the German Research Foundation (DFG) under grant SCHU 2458/4-1. A preliminary version of this article appeared in the proceedings of Graph Drawing & Network Visualization 2015.

E-mail addresses: alex.igamberdiev@gmail.com (Alexander Igamberdiev) w.meulemans @tue.nl (Wouter Meulemans) andre.schulz@fernuni-hagen.de (André Schulz)

1 Introduction

To assess the quality of network visualizations, many criteria have been investigated, such as crossing minimization, bend minimization and angular resolution (see [12] 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 [11]: 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 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, 11]. 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: each of these convex-hull vertices 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. [8] improve this by giving an algorithm that uses $n/2 + 4$ segments. Moreover, this algorithm 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 Section 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.

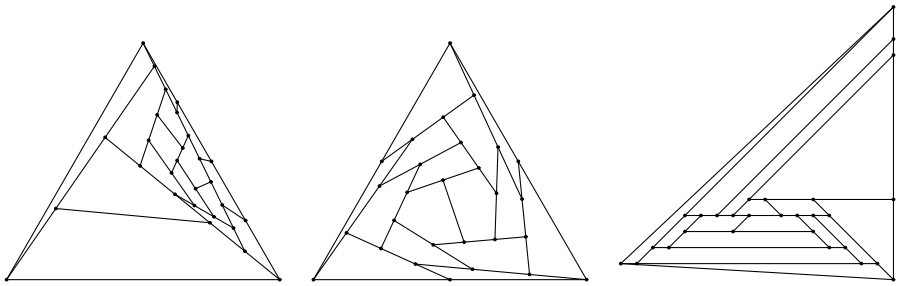

Figure 1: Result of the various algorithms for the same graph and outer face. (left) Reconstruction algorithm. (middle) Windmill algorithm. (right) Mondal algorithm.

Contributions. We present two different new $O(n^2)$-time algorithms (Section 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 Figure 1. We run several experiments and evaluate the performance of these algorithms by measuring geometric 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[1].

2 The Reconstruction Algorithm

The Reconstruction algorithm is based on an operation on embedded graphs that we call *edge insertion*[2]: 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. Its inverse is *edge removal*: pick an edge (u, v), remove it, and make the series reduction on u and v. These operations are illustrated in

[1] `http://reference.wolfram.com/language/ref/GraphData.html`

[2] This is *not* the graph-theoretic notion of edge insertion.

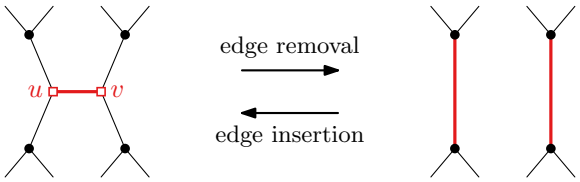

Figure 2: Edge removal (left to right) is the inverse of edge insertion (right to left). It reduces the number of vertices by 2 and the number of edges by 3.

Figure 2. The algorithm consists of two phases: in the first it deconstructs the desired graph, finding a sequence of edge removals, until the triangular prism is obtained; in the second we reverse this sequence of edge removals, and reconstruct the graph while maintaining a flat-angle drawing.

2.1 Deconstruction with Edge Removals

In this first phase of the algorithm we deconstruct the given graph, computing a sequence of edge removals to obtain the triangular prism. That is, we repeatedly perform an edge removal, until we obtain a triangular prism in one of its two planar embeddings. The existence of such a sequence is readily implied by folklore stating that every cubic 3-connected graph can be obtained from K_4 by a sequence of edge insertions (e.g, [5, page 243]). However, for our purpose we need a slightly stronger version, as our reconstruction phase (next section) cannot cope with edge insertions in the outer face.

While edge insertion preserves 3-connectivity, planarity and 3-regularity, edge removal can break these properties and produce double edges. Hence, we call an edge a *removable edge* if its removal maintains planarity, 3-regularity and 3-connectivity. We may remove only such removable edges in the deconstruction process; we implicitly suppose that edge removal is applied only to removable edges. Characterizing removable edges is easiest through the dual graph, as shown with the lemma below.

Lemma 1 *An edge is removable if and only if its dual is not part of a separating triangle in the dual graph.*

Proof: The operation on the dual graph that corresponds to edge removal on the primal graph is the *edge contraction*, as illustrated in Figure 3.

Graph G is cubic, planar and 3-connected if and only if its dual G^* is a planar triangulation. The lemma is thus equivalent to stating that the dual remains a planar triangulation, when contracting a dual edge that is not part of a separating triangle.

This is indeed trivial: edge contraction can be executed preserving the planar embedding and every face stays triangular. We have to ensure only that no double edges appear; this is exactly guaranteed by contracting only edges that are not part of a separating triangle. □

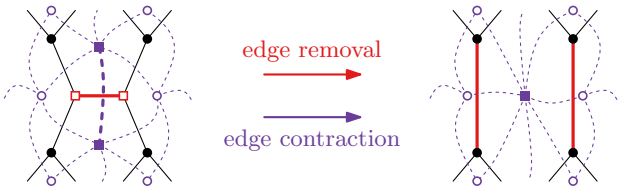

Figure 3: Dual operations: edge removal and edge contraction. The primal graph uses solid lines and its thick red edge is being removed. The dual graph is dashed and its thick edge is being contracted.

We are now ready to formulate and proof the deconstruction process.

Theorem 1 *Any cubic planar 3-connected graph G that is not K_4 with an (arbitrarily) fixed outer face f can be deconstructed to the graph of a triangular prism by a sequence of edge removals such that no edge is removed from f.*

Proof: The proof is by induction on the number of edges (which must be a multiple of 3 in any cubic graph). We do not consider K_4: the base is a graph with 9 edges. There is only one such graph, the triangular prism.

For the induction step we need to show that every planar cubic 3-connected graph G with more than 9 edges has a removable edge not on the fixed outer face f. Or, equivalently on the dual graph, every planar triangulation G^* with more than 9 edges has an edge that is not part of a separating triangle (Lemma 1) and is not incident to vertex f^*, the dual of the fixed outer face.

If G^* contains no separating triangle, then any edge not incident to f^* can be selected. Otherwise, consider a separating triangle T. It splits G^* into two components; at least one of them does not contain f^*. Pick as the interior of T one of these components that does not contain f^* . Let T' be a minimal separating triangle—containing no other separating triangle—contained by T; if T contains no separating triangle, we simply use T as T'. By our choice of the interior of T, f^* is either a vertex of T' or lies outside T'. Pick a vertex of T' that is not f^*. As G^* is a planar triangulation, this vertex has an edge to a vertex inside T'. By construction, this edge is not part of a separating triangle nor is it incident to f^*.

Thus, we can always find a removable edge not on the outer face, remove it and use the induction hypothesis. This finishes the proof of Theorem 1. □

Corollary 1 *Any cubic planar 3-connected graph G that is not K_4 with an (arbitrarily) fixed outer face f can be constructed from the graph of a triangular prism by a sequence of edge insertions such that no edge is inserted into f.*

2.2 Reconstruction with Edge Insertions

Using the techniques of the previous section, we have found a sequence of edge insertions from one of the two planar embeddings of the triangular prism that

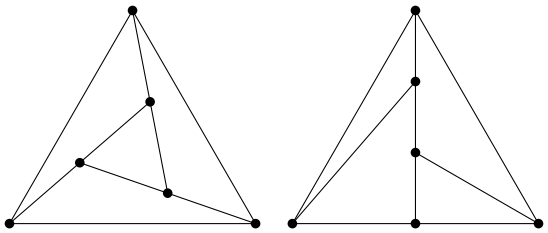

Figure 4: The triangular prism admits to planar embeddings, both of which have a simple flat-angle drawing.

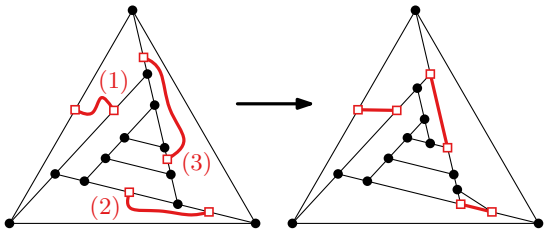

Figure 5: Edge insertion that connects (1) noncollinear edges of a face, (2) collinear edges separated by one vertex, and (3) collinear edges separated by two or more vertices. In case 2 and 3, we reassign the flat angle at the first and/or last separating vertex.

leads to the given graph. With this sequence, we reconstruct the original graph in such a way that we obtain a flat-angle drawing. The embedding of the triangular prism either has three or four vertices on the outer face. Both cases have a trivial flat-angle drawing (see Figure 4) and we start the reconstruction from this drawing. We then apply the edge insertions while maintaining a flat-angle drawing (see Figure 5). 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 (such as (1) in Figure 5), 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. This maintains a plane flat-angle drawing as faces must be nonstrictly convex.

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 Figure 6(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 Figure 6(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

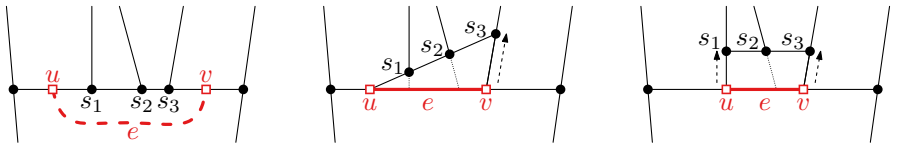

Figure 6: (a) Inserting edge $e = (u, v)$ with its endpoints on the same side of a face (a). (b) The standard insertion. (c) An alternative strategy.

alternative method for inserting (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 place u at the old position of s_1, and v at the old position of s_k as depicted in Figure 6(c). We refer to the latter strategy as the *alternate insertion operation*.

2.3 Variants

The deconstruction sequence, and hence the result of the construction, is not necessarily unique. For the later analysis (Section 5), we distinguish three different strategies on how to select the removed edge from the set of all removable edges that are not incident to the fixed outer face:

(R) We select an edge randomly.

(S) We select an edge with minimal sum of the degrees of its incident faces.

(L) Analogous to (S), we select an edge with maximal sum of degrees.

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.

With three different deconstruction strategies (R, S and L) and an alternative insertion operation (ALT), we have six variants of the Reconstruction algorithm, referred to as REC-R, REC-R-ALT, REC-S, REC-S-ALT, REC-L and REC-L-ALT. In the -ALT variants, the alternative insertion operation is applied whenever possible; in the other variants it is never used.

3 The Windmill Algorithm

In contrast to the Reconstruction algorithm and Mondal's algorithm—which build a drawing bottom-up using local structures—the Windmill algorithm is recursive, taking a top-down approach by starting with the overall structure. It computes a flat-angle drawing, working its way inward from the outer face, until all vertices have been processed.

We have a planar cubic 3-connected graph G with at least 6 vertices. Some combinatorial planar embedding of G is given and three of its vertices on the

outer face are marked as suspension vertices. The algorithm constructs a flat-angle drawing $\delta(G)$; we use $\delta(\cdot)$ to refer to the embedded positions of vertices and geometries representing subgraphs of G. In addition to G, the recursion takes as input a simple cycle C in G. The interior and exterior of C are defined with respect to the given combinatorial embedding; we use *strict* interpretations of these: edges and vertices on C are considered neither inside nor outside. The algorithm poses the following precondition on C.

Precondition 1 *C is a simple cycle in G, such that $\delta(C)$ is the boundary of a nonstrictly convex polygon. A strictly convex vertex of $\delta(C)$ corresponds to either a suspension vertex or a vertex with an edge outside of C. A flat vertex of $\delta(C)$ corresponds to an internal vertex with an edge inside C.*

Hereafter, we simply use convex to refer to strictly convex vertices in a drawing of a cycle, $\delta(C)$. The *sides* of $\delta(C)$ are the line segments between two consecutive convex vertices. Any edge or flat vertex of C lies on precisely one side; any convex vertex lies on two sides. With F we denote the cyclic sequence of faces inside C as they occur along the boundary of C. Consecutive edges of C that are incident to the same inside face f give rise to only one occurrence of f in F, but multiple occurrences are possible; see Figure 7(c) for an example of such a face.

To initialize the algorithm, we use as C the outer face of G. To meet Precondition 1, we position the suspension vertices as the corners of an equilateral triangle; the other vertices of C are positioned equidistantly along its sides (Figure 7(a)).

A call to the windmill algorithm with cycle C computes a position $\delta(v)$ for each vertex inside C. The edges inside C are line segments connecting their endpoints. To obtain a flat-angle drawing, the algorithm places each vertex v in such a way that $\delta(v)$ is collinear with the placement of two of its neighbors. One recursive step consists of the execution of the first applicable of the four cases below, to be detailed and proven correct in the upcoming sections.

1. *At most one vertex*: there is at most one vertex strictly 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 Figure 7(b→c,e→f).

2. *Repeated face*: a face f occurs more than in F, splitting its interior. We draw its paths inside C as line segments and recurse on a subcycle for each path. See Figure 7(c→d)

3. *Adjacent faces*: two faces, f_1 and f_2, are adjacent but not consecutive in F, assuming that there are two different sides of $\delta(C)$ to which the faces are incident. We draw three line segments to represent the paths inside C along the two faces and recurse on the two subcycles created. See Figure 7(a→b).

4. *Windmill*: none of the above holds. We create a windmill pattern with the sequence of faces along C. We recurse on the cycle inside the windmill. See Figure 7(d→e).

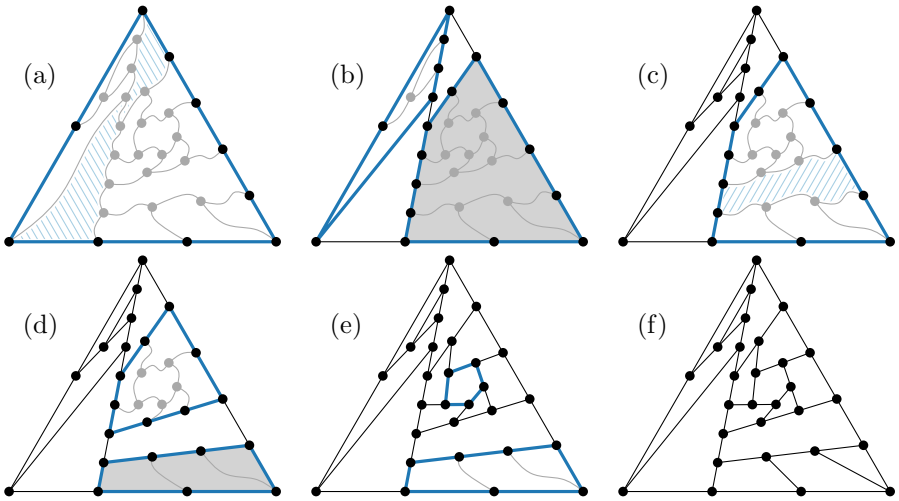

Figure 7: The Windmill algorithm. Cycles are drawn thick; unshaded cycles are processed in the next step. (a) Initial call. (b→ . . . →e) Consecutive states. (f) Final result. (e→f) Two cycles are processed. (a,c) Hashures indicate faces relevant for case 3 and 2.

Note the 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. Case 4 must handle such a pattern using additional recursive calls. Details are found in Section 3.4.

To prove that the algorithm is correct, we use induction on the number of edges inside C. The first case does not result in a recursive call and thus forms the base case for this induction. The other three cases result in one or more recursive calls, for which we shall argue that the number of edges inside the cycle is smaller, thus allowing us to apply the induction hypothesis.

Crucial to this proof 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.

In the upcoming sections, we discuss the precise conditions, constructions and correctness proofs for each of these cases. We also use the dual G^* of graph G. Note that G^* is a maximal triangulation. With C^* we denote the subgraph of G^* restricted to faces interior to C.

3.1 At most one vertex

Condition. Cycle C has at most one interior vertex v.

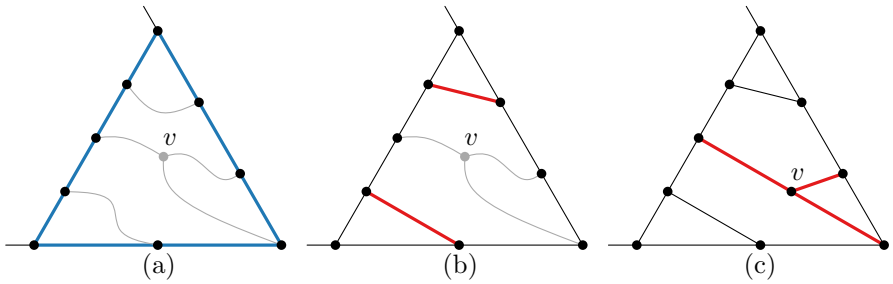

Figure 8: (a) Cycle C (blue) with one vertex inside. (b) Any chords are drawn as line segments (red). (c) One line segment for two edges of v is chosen such that it does not coincide with C; the third edge is drawn as a line segment to the position for v.

Construction. Cycle C may have a number of chords. Before dealing with v, each chord $e = (u, w)$ is drawn as a line segment between $\delta(u)$ and $\delta(w)$ (Figure 8).

If v is present, we proceed as follows. Vertex v has three neighbors on C. By 3-connectivity, not all three neighbors of v can lie on the same side of $\delta(C)$. Hence, we can pick a pair of neighbors of v such that the line segment connecting them lies strictly inside $\delta(C)$. We place v halfway this line segment and simply connect the third neighbor.

Proof. We must prove that this results in a plane drawing inside C. Since G is a planar graph, we know that no two chords intersect. Moreover, a chord cannot have its endpoints on one side of $\delta(C)$, argued as follows. If there are no other vertices on C in between the endpoints, the chord would imply a multi-edge. Any vertices in between must be flat vertices and thus have an edge inside C by Precondition 1. This implies that the endpoints of the chord make a separating pair of vertices contradicting the assumption that G is 3-connected.

What remains is to argue that our placement of v (if present) is correct. Consider the state of the drawing after inserting the chords but before placing v. The chords partition the interior of $\delta(C)$ into a number of convex polygons. By planarity, v lies inside one of these polygons and its neighbors are placed on its boundary. By construction, v is placed strictly inside the polygon, resulting in a planar drawing.

3.2 Repeated face

Condition. Cycle C contains a face f that occurs more than once along C, splitting the subgraph. In terms of the dual, f^* is a separating vertex for C^*.

Construction. Consider the maximal components C_i^* that arise when removing f^* from C^*. There are at least two such components. The cut edges between C_i^* and its complement in G^* define a simple cycle C_i in the primal graph. This partitions C in an even number paths, alternatingly along f and along cycle C_i.

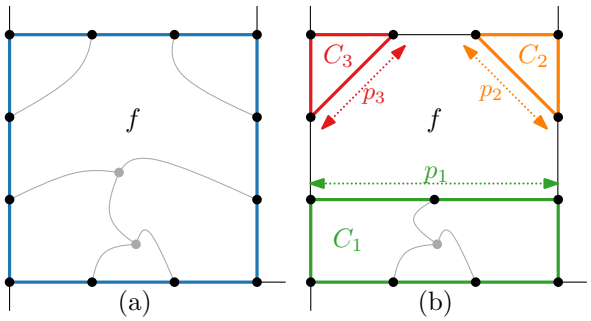

Figure 9: (a) Cycle C (blue) with a face f occurring multiple times along C. (b) Paths p_i in C are drawn as line segments and result in smaller cycles for recursion.

Each cycle C_i has precisely two vertices, u_i and v_i, on C that are incident to f. These vertices partition C_i into two paths: the chordal path p_i along f and the path q_i that follows C. We draw a single line segment for p_i between $\delta(u_i)$ and $\delta(v_i)$ and place the vertices of p_i in order equidistantly along this line segment. We then recurse on C_i. This is illustrated in Figure 9.

Proof. Each chordal path p_i is drawn as a line segment in the convex polygon $\delta(C)$. By planarity of G, these line segments cannot intersect each other.

The line segment $\delta(p_i)$ must lie strictly inside $\delta(C)$, as 3-connectivity with Precondition 1 implies that u_i and v_i lie on different sides of $\delta(C)$. This is analogous to the argument made for chords in the previous case.

We must prove that $\delta(C_i)$ meets Precondition 1. Except for u_i and v_i, any vertex on q_i—the path that follows C—has an edge inside C_i if and only if it has an edge inside C: Precondition 1 on C implies that it also holds for C_i. Except for u_i and v_i, any vertex on p_i—the chordal path—has an edge inside C_i and cannot be a suspension vertex: these vertices are flat on $\delta(C_i)$ as they lie along the line segment $\delta(p_i)$. Vertices u_i and v_i have their third edge outside C_i by construction. These vertices must be convex on $\delta(C_i)$: the interior angle must be strictly smaller than that in $\delta(C)$, which is at most π as $\delta(C)$ is convex.

The recursion on C_i entails a reduction in the number of interior edges, since any edge enclosed by C_i must be enclosed by C and the edges on the chordal path p_i are interior to C but not to C_i.

3.3 Adjacent faces

Condition. There is a pair of faces, f_1 and f_2, that share an edge e but do not occur consecutively along C. In terms of the dual, f_1^* and f_2^* are a separating vertex pair for C^*. Let u_i and v_i denote the first and last vertex of f_i along C in clockwise order, for $i \in \{1, 2\}$. This is illustrated in Figure 10(a). We require that either or both u_1 and u_2 or v_1 and v_2 lie on different sides of $\delta(C)$. If this is not the case, it is handled by the windmill case described in the next section.

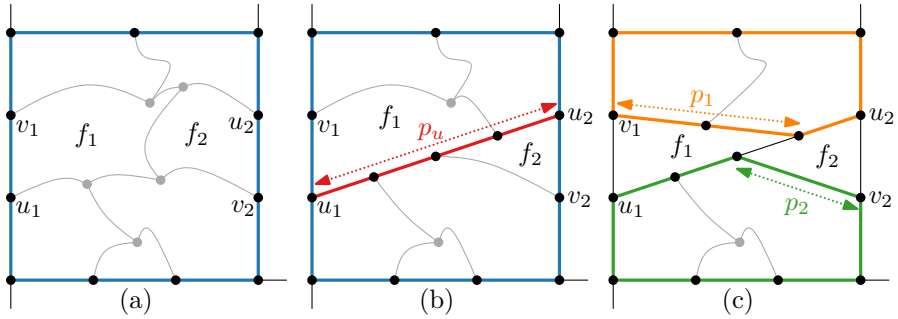

Figure 10: (a) Cycle C (blue) with adjacent faces, f_1 and f_2, that are nonconsecutive along C. (b) First, we place path p_u as a line segment. (c) We draw p_1 and p_2 as line segments and obtain two cycles for recursion.

Construction. Consider the path p_u inside C along f_1 and f_2 from u_1 to u_2, and similarly, p_v from v_1 to v_2. Both paths include the shared edge e. By our assumption, at least one of these paths ends on different sides of $\delta(C)$. Without loss of generality, assume it is p_u.

We draw p_u as a line segment between $\delta(u_1)$ and $\delta(u_2)$, placing its vertices equidistantly. We split p_v into p_1 and p_2 by removing edge e, using the index to denote along which face the paths lies. We draw p_1 as a line segment between $\delta(v_1)$ and its endpoint of e, as given by $\delta(p_u)$, placing its vertices equidistantly. We handle p_2 analogously. This is illustrated in Figure 10(b-c).

Removing f_1^* and f_2^* from C^* yields two maximal components in the dual graph. Each defines a simple cycle in the primal graph upon which to recurse.

Proof. Path p_u is drawn as a chordal line segment inside $\delta(C)$. Due to our assumption, it starts and ends on different sides: it cannot overlap a side of $\delta(C)$. This creates two convex polygons, in which we draw paths p_1 and p_2 as chordal line segments, one in each polygon. Thus, they cannot intersect or coincide with $\delta(C)$.

We prove as follows that a recursive cycle C' meets Precondition 1. Analogous to the repeated-face case, any vertex that originates from C is correctly positioned on $\delta(C')$ in terms of convexity. Any vertex on C' interior to C is adjacent to f_1 or f_2. If it is not adjacent to both, it must have its third edge interior to C'. By construction, it is and must be flat on $\delta(C')$.

The endpoints of e are incident to both f_1 and f_2. As e is outside both recursive cycles, these must be convex in the drawing. Both f_1 and f_2 are outside the recursive cycle, and one of them makes a flat angle at the endpoint of e. Thus, their combined angle is strictly larger than π, implying an interior angle strictly less than π for the recursive cycle.

To show correct recursion, observe again that any edge interior to one of the recursive cycles is interior to C, but at least five edges interior to C are incident to one of the shared faces and thus not interior to one of the recursive cycles.

3.4 Windmill

Condition. No other condition holds: C contains at least two vertices; any face along C occurs exactly once; two nonconsecutive faces along C share an edge only if they are incident to one side of $\delta(C)$ (see Figure 11(a)).

Construction. Consider the dual C^*, but restrict it to the faces along C; let us call this F^*. Clearly, F^* is an outer-planar graph. By the condition, we know that any chords in F^* can be only between two primal faces (dual vertices) that are incident to the same side of C. Thus, we may orient F^* in clockwise direction and find the shortest cycle W^* in F^* (see Figure 11(b)). This cycle can be found in a greedy way by starting from a dual vertex f^* on F^* such that the first vertex of face f on C is a suspension vertex or f has at least two edges in C (implying a convex vertex on $\delta(C)$).

We use cycle $W^* = \langle f_1^*, \ldots, f_k^* \rangle$ to create a windmill inside $\delta(C)$ as follows (refer to Figure 11(c-e) for illustrations). For each f_i^* on W^*, we compute the

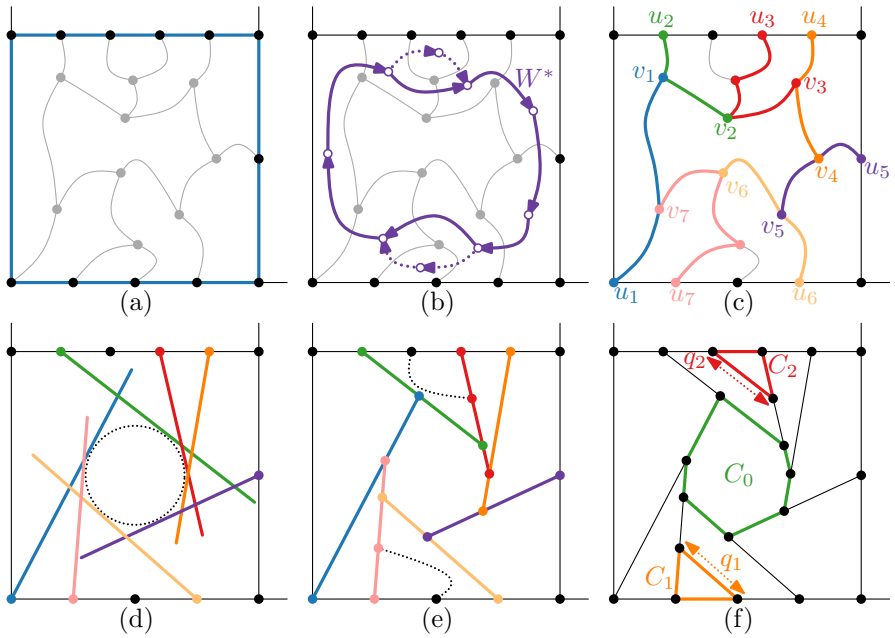

Figure 11: (a) Cycle C (blue) for which none of the other cases apply. There may be pairs of nonconsecutive adjacent faces: in this case, one on the top and one on the bottom side. (b) The (directed) dual F^* and its shortest cycle W^* are given in purple. (c) Using W^* we define 7 paths p_i from u_i to v_i, such that u_i lies on C and v_i on p_{i+1}. (d) We construct the left tangent to some circle c (dashed). (e) With these tangents we construct the windmill. We ignored two parts in this construction (dashed) due to chords in F^*. (d) These are added as triangles, resulting in three cycles for recursion.

primal path p_i starting at the first vertex u_i in clockwise direction of f_i on C to the first vertex v_i incident to f_{i+1} in counterclockwise direction along f_i. Now, we have paths $p_i = \langle u_i, \ldots, v_i \rangle$ such that u_i lies on C and v_i is a nonendpoint vertex of p_{i+1}. To create the windmill, we must define an appropriate line segment for each path p_i. To this end, consider some small circle c somewhere inside the convex hull of the u_i vertices. With t_i, we denote the left tangent from u_i on c. The line segment we draw for p_i starts at $\delta(u_i)$ and ends at the intersection between t_i and t_{i+1}. After placing these line segments, we position the vertices on p_i in before v_{i-1} equidistantly between $\delta(u_i)$ and $\delta(v_{i-1})$; we place the vertices on p_i after v_{i-1} equidistantly between $\delta(v_{i-1})$ and $\delta(v_i)$.

Consider removing W^* from C^*. This splits the dual subgraph into a number of maximal components, $C_0^*, \ldots, C_m^*$, where m is the number of chords used by W^*. Each C_j^* has a primal outline corresponding to a simple cycle C_j. One component is incident to all f_i in W^* and corresponds to the interior of the windmill. We assume that this is C_0^*. All other components C_j^* for $j \in \{1, \ldots, m\}$ correspond to subgraphs that are separated by a chord in F^* from the windmill.

Cycle C_0 corresponds to the cycle of the windmill. Its placement $\delta(C_0)$ is given directly by the construction. Thus, we recurse on C_0.

Any other component C_j^* was separated by a chord in F^*. Assume this chord is from f_i^* to f_{i+1}^*. C_j consists of three parts: edges on C, edges incident to f_i and edges incident to f_{i+1}. The edges on C and incident to f_{i+1} (part of p_{i+1}) have already been placed. What remains is to place a line segment to represent the edges incident to f_i. Let us call this path q_j. Path q_j has one endpoint on C and one on p_{i+1}. These have already obtained a placement, prescribed by $\delta(C)$ and the windmill respectively. The vertices on q_j are placed equidistantly between these endpoints. This creates a triangular $\delta(C_j)$; we recurse on each C_j (see Figure 11(f)).

Rather than using a clockwise order, we may also orient F^* in counterclockwise direction, reversing W^*. If we aim for a uniformly weighted harmonic system, we can also solve the local harmonic system with appropriate weights for constructing the windmill, to directly compute the final layout and avoid the need for solving the entire harmonic system after the windmill algorithm.

Proof. To see why this operation yields a planar drawing, we observe that the endpoints u_i are in (nonstrict) convex position. Hence, the tangents t_i occur in order along the circle c and the windmill is planar. Now, for each additional component C_j^*, we place another line segment. We observe that this occurs at most once for any f_i^* on W^*. Before placing the chord, face f_i lies in a convex polygon: it uses part of $\delta(C)$ which is convex, and two line segments placed for p_i and p_{i+1}. The line segment for q_j is a chord between different sides of this polygon. Thus, it causes no intersections.

We must argue why component C_0^* inside the windmill exists. As F^* does not have multi-edges, W^* has at least length three. If W^* has length greater than three, C_0^* must contain a dual vertex, as G^* is a maximal planar graph. If W^* has length three, but does not enclose a dual vertex, then the cycle of faces

meet at a single vertex v in the primal graph. If C is the outer face of G (i.e., this is the initial call to the windmill algorithm), then G must be K_4, violating our assumption that G has at least 6 vertices. If C is not the outer face of G, at least two of the faces on W^* have an edge outside of C (by 3-connectivity) and thus be incident to more than one side of C. As there is at least one more vertex inside C, there must be a pair of faces on W^* that are nonconsecutive on F^* (but adjacent as they are on W^*). Such a pair cannot be incident to a unique face, violating the condition of this windmill case: the adjacent-faces case should be applied instead.

To see why Precondition 1 holds for $\delta(C_0)$, first observe that it can have no suspension vertices. Any vertex on C_0 with an edge outside C_0 is convex on $\delta(C_0)$ as it is the intersection between two consecutive tangents used in the construction. Any other vertex on C_0 must have its edge inside C_0. As it is not the endpoint of some p_i, it has been placed equidistantly between some v_i and v_{i+1}, thus implying it is flat on $\delta(C_0)$.

For each $\delta(C_j)$ with $j > 0$, we must argue the same. Again, assume this component was caused by a chord in W^* from f_i^* to f_{i+1}^*. As f_i and f_{i+1} lie on the same side of C and are exterior to C_j, C_j cannot have suspension vertices. $\delta(C_j)$ has three convex vertices. For each we argue that it has two faces outside C_j and thus an edge outside of C_j. This implies that it should indeed be convex for recursion on C_j.

1. The endpoint u_{i+1} of p_{i+1}. u_{i+1} is incident to f_{i+1} and the face outside of C; both are exterior to C_j
2. The endpoint of q_j on C. This endpoint is incident to f_i and the face outside of C; both are exterior to C_j.
3. The endpoint of q_j on p_{i+1}. This endpoint is incident to f_i and f_{i+1}; both are exterior to C_j.

All other vertices on C_j are flat and hence must have its edge inside C_j. For this, observe that all vertices also on C are on the same side (flat on $\delta(C)$) and hence have their third edge inside C_j. Vertices incident to f_i or f_{i+1} cannot have an edge exterior to C_j, as it would imply that C_j^* is not a maximal component.

For each recursion, any edge interior to the recursion cycle is interior to C, but each cycle does not include at least three edges that are part of the windmill cycle C_0. Thus any recursive cycle has less edges in its interior than C, allowing the induction hypothesis to be applied.

3.5 Variants

The Windmill algorithm leaves little room for extensive strategies, since the operation's order is crucial to its correctness. However, in performing the Windmill operation, we can choose either 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.

4 The Mondal Algorithm

Mondal et al. [8] describe two linear-time algorithms for drawing cubic planar 3-connected graphs: one results in a grid drawing with $n/2 + 4$ line segments with six slopes; the other attains minimal visual complexity with seven slopes but does not produce a grid drawing. Both algorithms introduce the vertices as given by a canonical order.

We observe that the grid algorithm as described by Mondal et al. [8] is flawed. The example in Figure 12 illustrates the problem. When adding a chain of vertices from u to v (case 4d in [8], see Figure 12(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 [8], see Figure 12(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 Figure 12(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

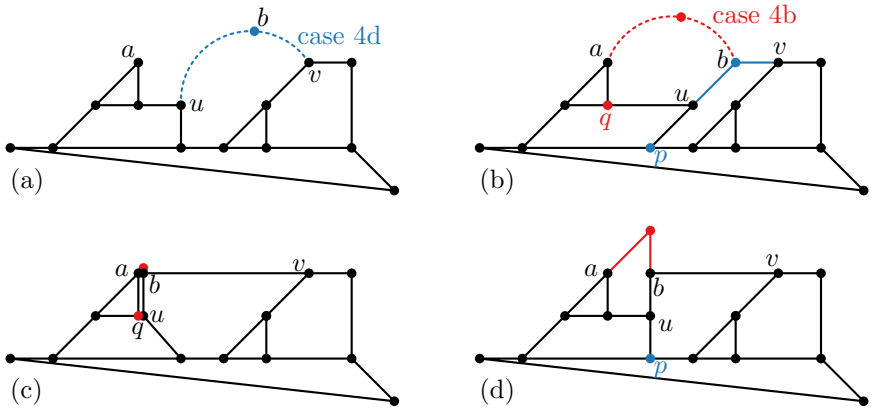

Figure 12: (a) State before adding b between u and v. (b) After adding b, before adding a vertex between a and b. (c) Rotating about q as described in [8] results in a nonplane drawing. (d) Rotating about p as described here yields a plane result.

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 (or $n/2 + 4$ for MON-GRID). The drawings (Figure 1) are obviously different, but—as the visual complexity is the same—we need other 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 Data Sets

We generated all planar cubic 3-connected graphs with 24, 26, 28 and 30 vertices, using *plantri*[3]. From each batch we sampled 500 graphs uniformly at random without duplicates, resulting in a total of 2000 graphs. This creates the Random data set.

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

To further validate our results, we collected graphs with more than 30 vertices. However, the above graph-selection procedure becomes infeasible. Instead, we used *PlanarMap*[4] to directly generate 500 random planar cubic 3-connected graphs, for each (even) size of 32 up to and including 50 vertices. Note that these randomly generated maps are independent and hence have a (very) small probability of containing duplicates. The graphs with 32 to 40 nodes constitute the Random Medium data set; the graphs with 42 to 50 nodes the Random Large data set. Both these data sets hence contain 2500 graphs.

Finally, we also consider a special graph collection, in particular, the one found in Wolfram Mathematica. From this collection, we extracted the contained planar cubic 3-connected graphs with 6 to 30 vertices, and used these 146 graphs as a data set. This constitutes the Wolfram data set. To analyze for a structural bias in this data set, we also performed a random sampling of all cubic planar 3-connected graphs (using the *plantri* approach), but now such that we obtain the same distribution of graph sizes as in the Wolfram data set. We refer to these graphs as the Pseudo-Wolfram data set.

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. Edge lengths should neither be too short nor too long. Informal investigation of minimal edge lengths suggested only tiny differences, though MON-GRID was slightly ahead of the other algorithms. Hence, we in particular look at avoiding long edges in the remainder: we consider lower values for edge length to be better.

Face 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.

[4] `http://www.lix.polytechnique.fr/~schaeffe/PagesWeb/PlanarMap/index-en.html`; this implementation combines the results in [9, 10].

5.3 Algorithm Comparison: Random Data Set

Let us first consider the Random data set. Figure 13 shows the measured results for all graphs in the data set, summarized as a box plot. Below, we discuss these results per measure in more detail.

Angular resolution. The MON algorithms clearly perform better than the WIN algorithms, which in turn outperform the REC algorithms. This was to be expected due to the fixed slopes used in the MON algorithms.

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 REC algorithms. Though statistically significant (later in this section), the differences are only minor. The maximum edge length for the WIN and REC 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 REC 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 REC outperforms the MON algorithms, and MON-GRID is actually slightly ahead of MON-MIN.

We conclude from the above that the WIN algorithms generally outperform REC 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 angular 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 Figure 14. 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 REC 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

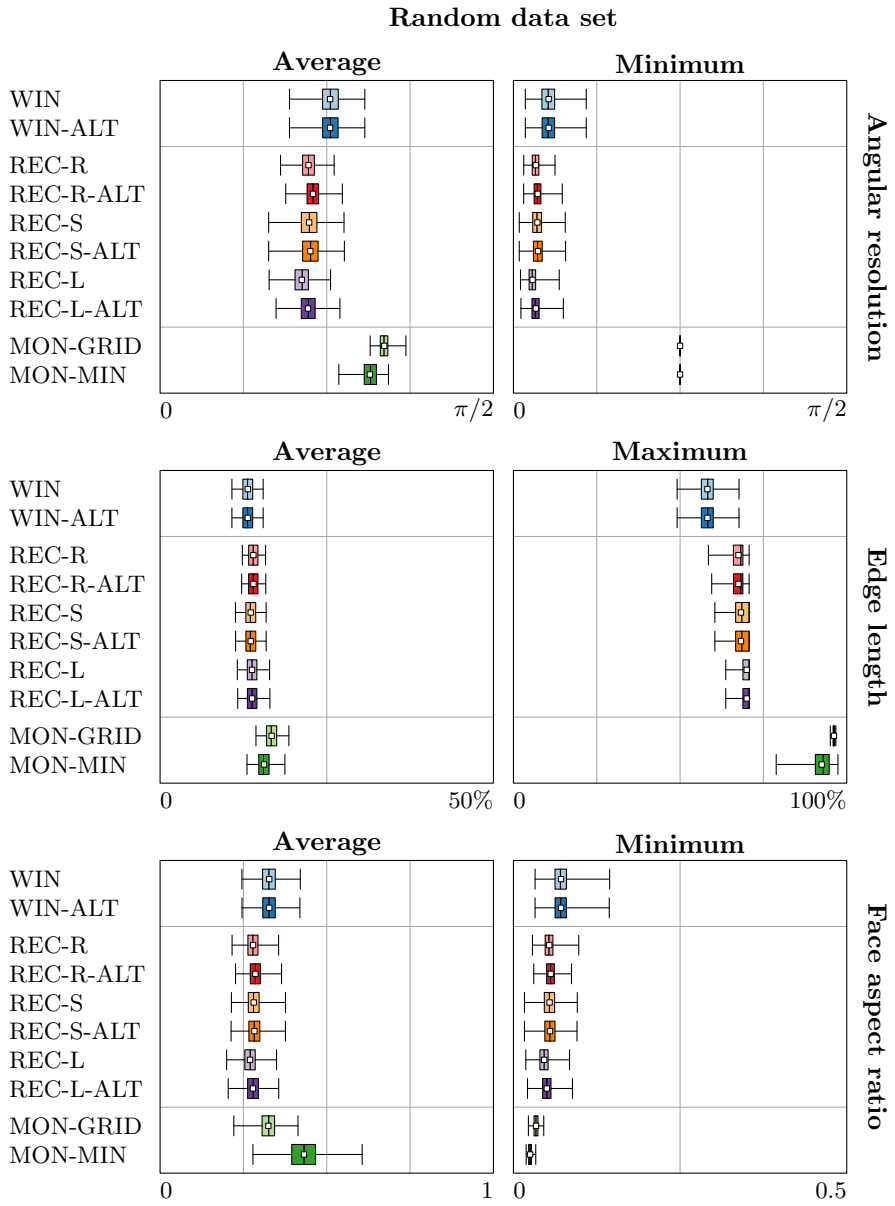

Figure 13: Box plot of the measured results for the Random data set. For length, lower values indicate better drawings; for the other measurements, higher values indicate better drawings.

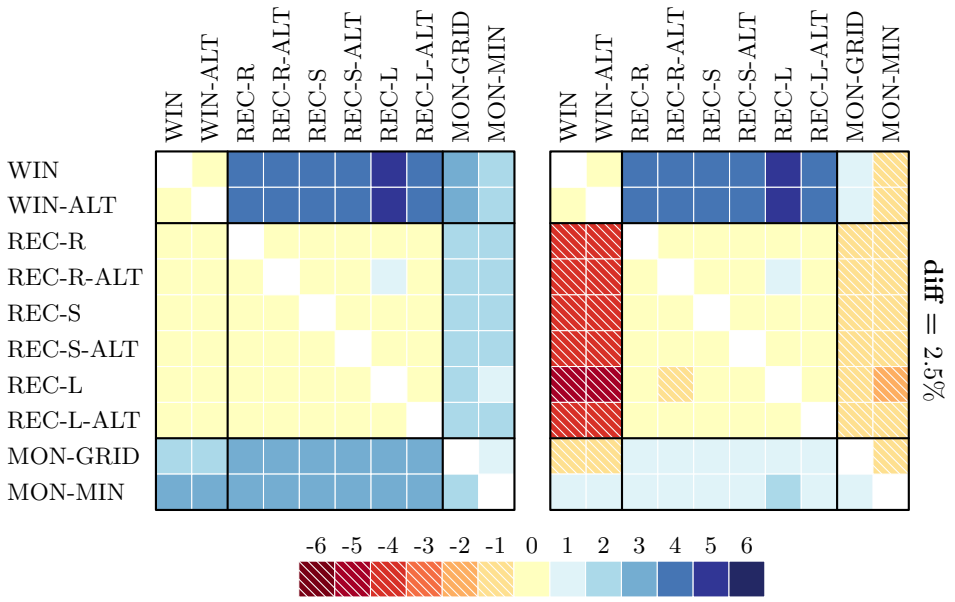

Figure 14: Analysis results for the Random data set. (Left) Number of "wins" (measures for which the row-algorithm outperforms the column-algorithm), using $p < 0.001$ in a Tukey HSD test with an estimated difference in means of at least 2.5%. (Right) The number of "wins" minus the number of "losses" in the left table, giving an overall view of relative performance.

"wins" more often than it "loses" compared to any other algorithm. However, the MON-GRID algorithm is outperformed by the WIN algorithms. The REC algorithms are not distinguishable between themselves. They outperform the MON algorithms for some measures, but are outperformed by the MON algorithms more often.

The right column of Figure 15 lists the charts reporting on any statistical significance (0%) and using a required estimated difference of 5%. The latter shows that the MON algorithms gain in performance: if a measure indicates a better performance for the MON algorithms, then the difference is more decisive. The former provides us some insight in the comparison of the REC algorithms, though we should take into account that the differences are minor. In particular, we may see some evidence to suggest that the ALT variants have a slight performance gain over the other variants. Moreover, the L strategy performs poorest. REC-R-ALT performs better than any other variant.

Larger graphs. We may repeat the above analyses on graphs with more vertices. To this end, we consider the Random Medium and Random Large data sets. Figure 16 illustrates the results of these two larger data sets together with the results obtained for the Random data set analyzed above.

In terms of angular resolution, the performance of the WIN and REC algo-

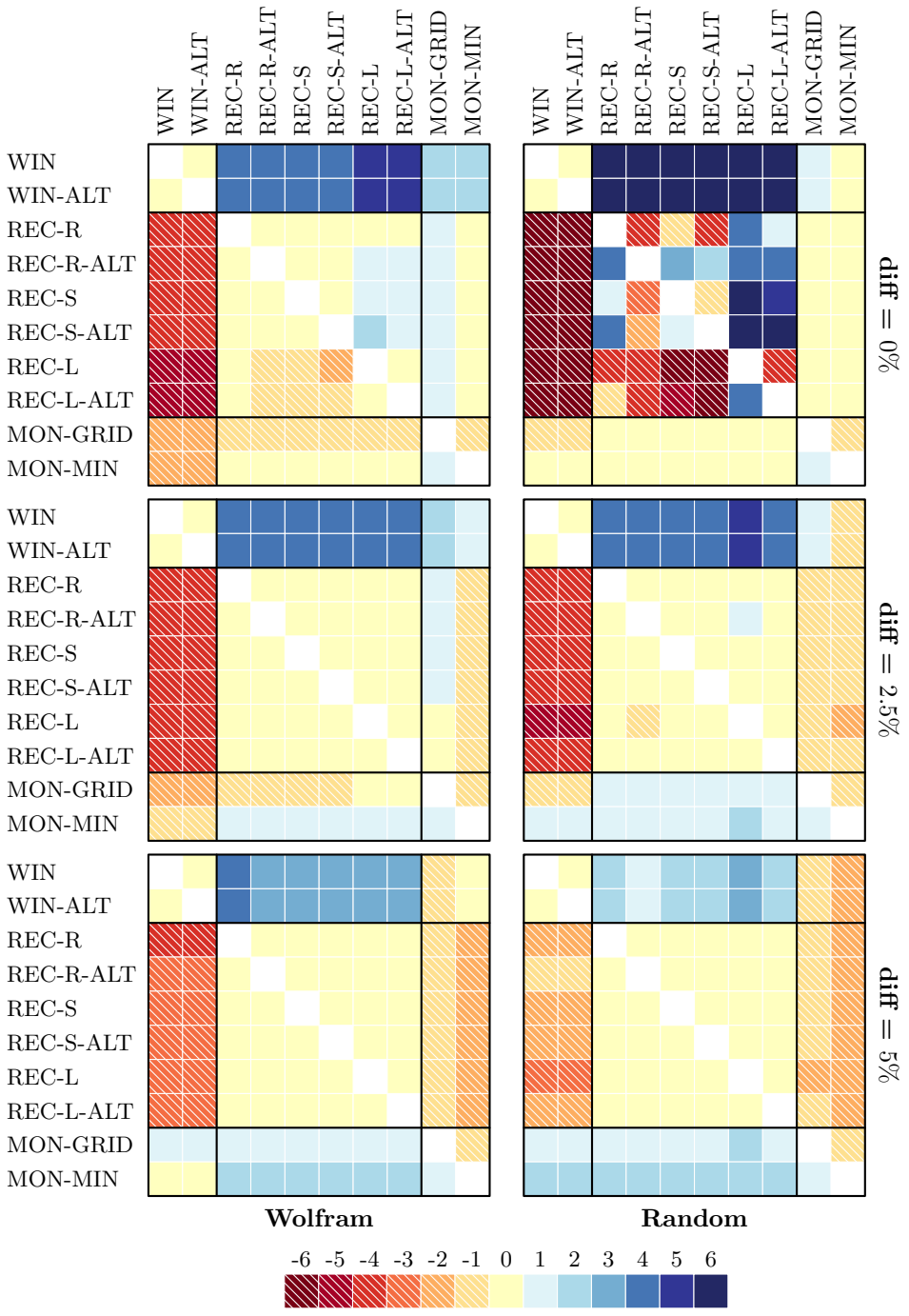

Figure 15: Comparison of algorithm performance for the Wolfram and Random data set, using $p < 0.001$ in a Tukey HSD test, using three levels of difference in means: 0%, 2.5% and 5%. Shown are the number of "wins" minus the number of "losses", giving an overall view of relative performance.

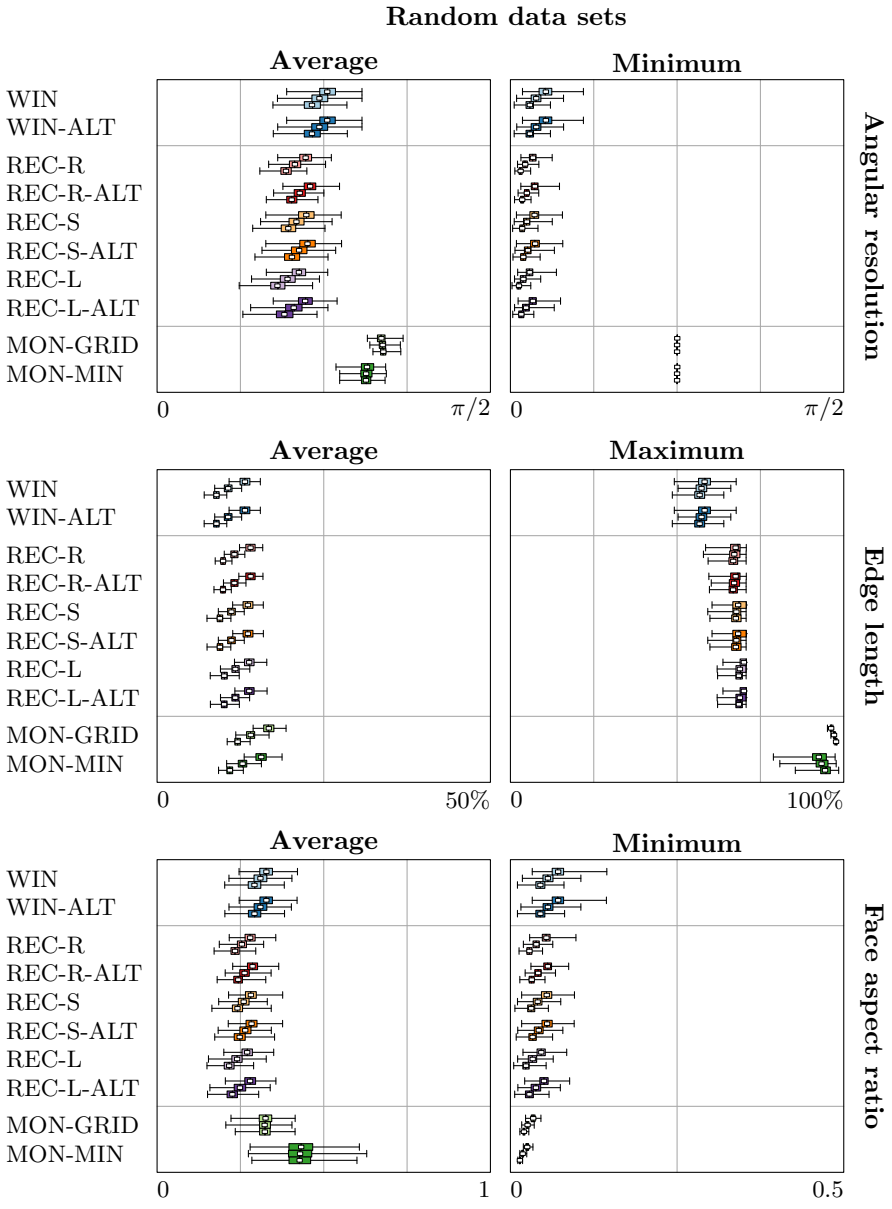

Figure 16: Three data sets are visualized concurrently. For each algorithm-criterion combination, we present three box plots to show results: the Random data set (top), the Random Medium data set (middle) and the Random Large data set (bottom). This allows us to spot trends on how these criteria vary as we increase the graph size. For length, lower values indicate better drawings; for the other measurements, higher values indicate better drawings.

rithms slightly decreases, though the MON algorithms stay at roughly the same level of performance. Though the MON algorithms already outperformed the other algorithms in this criterion, the difference becomes increasingly noticeable for larger graphs.

The average edge length decreases for all algorithms, simply due to the larger number of edges that need to be drawn. Though the WIN and REC algorithms maintain lower average edge length, their lead on the MON algorithm decreases even further. In terms of maximal edge length, we observe a very minor decrease in the WIN and REC algorithms; this is likely caused by the larger number of vertices that may be placed on the outer face. For the MON algorithm, there is a slight increase in the maximal edge length. Thus for maximal edge lengths, the difference between the WIN and REC algorithms and the MON algorithm increases, in favor of the former.

For face aspect ratio, the MON algorithms roughly maintain their average value, whereas the other algorithms decrease; the difference hence increases in favor for the MON algorithms. The minimum face aspect ratio decreases for all algorithms. The difference between the MON algorithms and the WIN and REC algorithms becomes smaller for larger graphs.

To summarize, the MON algorithms remain stable in terms of angular resolution (both average and minimum) and average face aspect ratio whereas the others decrease performance; for average edge length and minimum face aspect ratio, the difference between the algorithms decreases; only for maximum edge length, the MON algorithms' performance decrease whereas the others increase. Overall, this suggests that the MON algorithm becomes increasingly preferable for larger graphs.

5.4 Algorithm Comparison: Wolfram Data Set

Let us now turn to the Wolfram data set. The measurements are summarized in Figure 17; the Tukey HSD test results are given in the left column of Figure 15. Roughly the same patterns can be observed as for the Random data set. As the graphs in this data set are typically smaller ($n = 16.5$ on average), some minor differences arise. In particular, the angular resolution tends to increase for the WIN and REC algorithms, compared to the values obtained for the Random data set. However, for the MON algorithms, there in fact seems to be a slight decrease. Moreover, the average face aspect ratio of the MON algorithms decreases: MON-MIN is in line with the REC algorithms and the MON-GRID algorithm has lost its lead on the WIN algorithms.

To investigate whether this data set has a structural bias, we repeat the analysis procedure for the pseudo-Wolfram data set and compare the results. The main difference may be observed at a minimal estimated difference in means of 5%, see Figure 18. We see that the WIN algorithms are only slightly ahead of the REC algorithms (outperforming in only one measure, rather than three to four) and slightly behind the MON algorithms for the pseudo-Wolfram data set. This suggests that there may be some bias towards a structure that is exploited by the WIN algorithms.

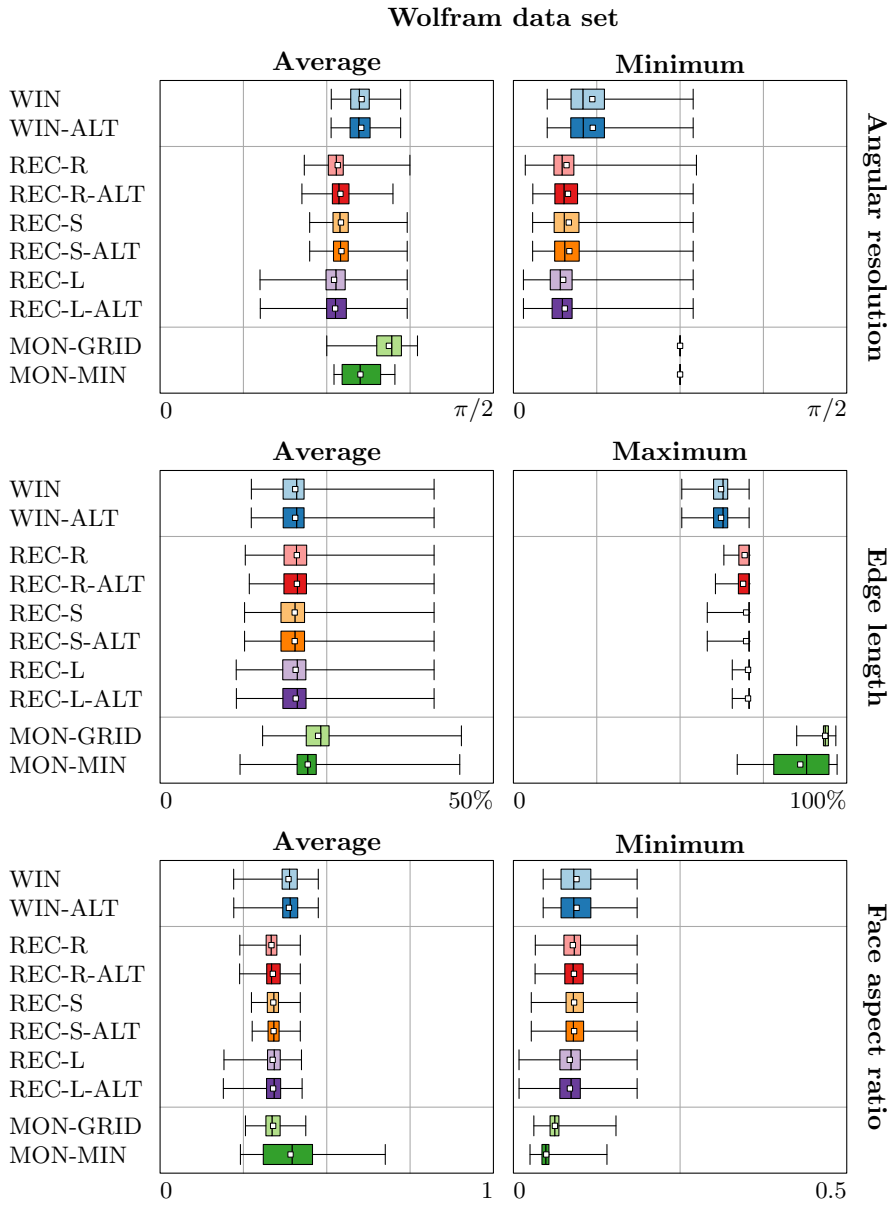

Figure 17: Box plot of the measured results for the Wolfram data set. For length, lower values indicate better drawings; for the other measurements, higher values indicate better drawings.

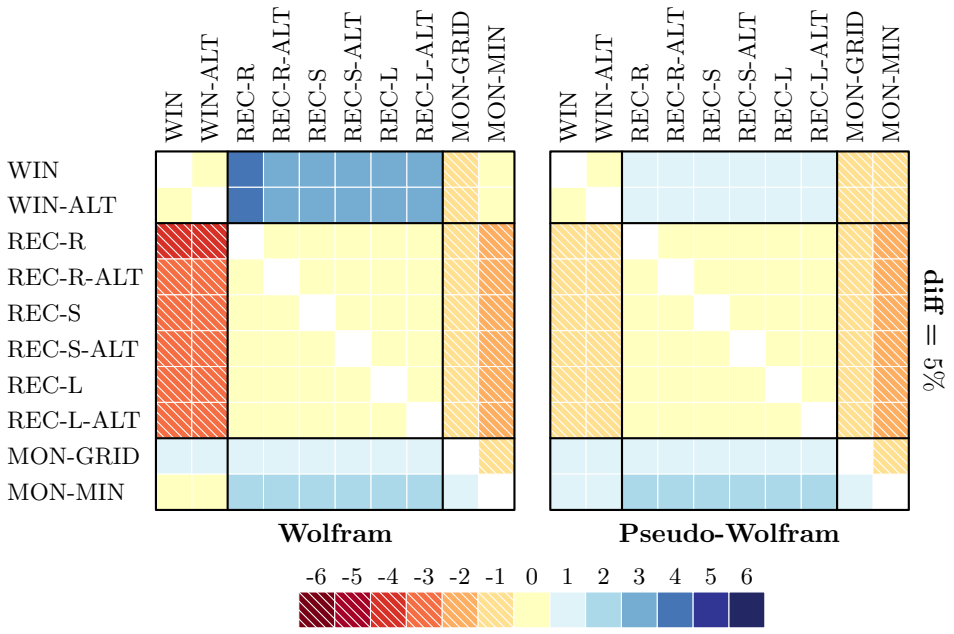

Figure 18: Comparison of algorithm performance for the Wolfram and the pseudo-Wolfram data sets. Shown are the number of "wins" minus the number of "losses", giving an overall view of relative performance.

6 Conclusions

We studied algorithms for drawing cubic planar 3-connected graphs with minimal visual complexity, that is, 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. [8] and argued that their variant that achieves the minimal visual complexity can result in a grid drawing with only six rather than seven slopes with minor modifications.

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 Reconstruction 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 [7, 11]). 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. Though it would break its provable properties such as good angular resolution due to fixed slopes and a bounded grid size, it may provide us further insight into the effect of the harmonic equations and the methods for generating the flat-angle drawing separately.

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. doi:10.1007/978-3-319-03841-4_11.

[2] V. Dujmović, D. Eppstein, M. Suderman, and D. Wood. Drawings of planar graphs with few slopes and segments. *Computational Geometry: Theory and Applications*, 38:194–212, 2007. doi:10.1016/j.comgeo.2006.09.002.

[3] S. Durocher and D. Mondal. Drawing plane triangulations with few segments. In *Proceedings of the 26th Canadian Conference on Computational Geometry (CCCG 2014)*, pages 40–45, 2014.

[4] S. Durocher, D. Mondal, R. Nishat, and S. Whitesides. A note on minimum-segment drawings of planar graphs. *Journal of Graph Algorithms and Applications*, 17(3):301–328, 2013. doi:10.7155/jgaa.00295.

[5] B. Grünbaum. *Convex Polytopes*. Graduate Texts in Mathematics. 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. *Computational Geometry: Theory and Applications*, 31(1–2):31–61, 2005. doi:10.1016/j.comgeo.2004.07.003.

[7] G. Hültenschmidt, P. Kindermann, W. Meulemans, and A. Schulz. Drawing trees and triangulations with few geometric primitives. In *Abstracts of the 32nd European Workshop on Computational Geomemetry (EuroCG 2016)*, pages 55–58, 2016.

[8] D. Mondal, R. Nishat, S. Biswas, and S. Rahman. Minimum-segment convex drawings of 3-connected cubic plane graphs. *Journal of Combinatorial Optimization*, 25:460–480, 2013. doi:10.1007/s10878-011-9390-6.

[9] D. Poulalhon and G. Schaeffer. A bijection for triangulations of a polygon with interior points and multiple edges. *Theoretical Computer Science*, 307(2):385–401, 2003. doi:10.1016/S0304-3975(03)00226-3.

[10] G. Schaeffer. Random sampling of large planar maps and convex polyhedra. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC 1999)*, pages 760–769, 1999. doi:10.1145/301250.301448.

[11] A. Schulz. Drawing graphs with few arcs. In A. Brandstädt, K. Jansen, and R. Reischuk, editors, *Graph-Theoretic Concepts in Computer Science*, LNCS 8165, pages 406–417, 2013. doi:10.1007/978-3-642-45043-3_35.

[12] R. Tamassia. *Handbook of Graph Drawing and Visualization*. CRC Press, 2013.